HOCHSCHULE BOCHUM

– FACHBEREICH MECHATRONIK UND MASCHINENBAU –

MASTERTHESIS

INVESTIGATION OF CAPSULE NETWORKS REGARDING THE EXPLAINABILITY OF SEARCH ENGINE RANKINGS

UNTERSUCHUNG VON KAPSELNETZEN ZUR ERKLÄRBARKEIT VON SUCHMASCHINENRANKINGS

Autor:

FELIZIA QUETSCHER

Matrikelnummer: 18105947

Studiengang: Mechatronik M.Sc.

Prüfer: Prof. Jörg Frochte

Zweitprüfer: Prof. Martin Potthast

Abgabedatum: 12.09.2021

ABSTRACT (DEUTSCH)

Suchmaschinenrankings sind für Suchmaschinenbetreiber und –nutzer von zentraler Bedeutung und daher Ziel zahlreicher Optimierungsansätze [1–3], die seit einiger Zeit durch Methoden der Künstlichen Intelligenz (KI) unterstützt werden. [4–6] Bei der Anwendung von KI-Methoden ist die Transparenz des verwendeten KI-Modells von zentraler Bedeutung. In Bezug auf Suchmaschinenrankings bedeutet diese Transparenz die Nachvollziehbarkeit der KI-generierten Trefferreihenfolgen. [7, 8]

Die Bildersuche und die inverse Bildersuche könnten von einer KI-gestützten Erkennung der Bildmotive profitieren. Zur Erkennung von Bildmotiven werden aktuell „Convolutional Neural Networks“ (CNN, „Faltende Neuronale Netze“) eingesetzt. [9–11] Mit einem CNN können die Bilder nach ihrer Relevanz für einen Begriff basierend auf der Sicherheit des CNN in einer Reihenfolge angeordnet werden. [12] Im Rahmen einer transparenten KI-Anwendung ist es von großem Interesse, den Einfluss verschiedener Bildbereiche auf die resultierende Sicherheit zu kennen und erklären zu können.

Verfahren wie „Grad-CAM“, „Guided Backpropagation“ oder der „LIME“-Ansatz bieten Einblicke in die Wahrnehmung des CNN. [13–15] Aus diesen Ansätzen zeigt sich, dass CNN dazu zu tendieren, andere Merkmale als der Mensch für eine Entscheidung zu nutzen.

Kapselnetze sind eine spezielle Modellarchitektur, über die das Erlernen menschlich verständlicher Mermale ermöglicht wird. [16] Diese basieren auf CNN und enthalten einen speziellen Bereich, die Kapseln, in denen Merkmale eines Objekts gespeichert werden. Diese Merkmale sind beispielsweise die Größe oder Position eines Objekts und somit für den Menschen nachvollziehbar.

In dieser Arbeit wird ein Kapselnetz in Tensorflow und Keras erstellt und auf zwölf Klassen handgeschriebener Buchstaben des EMNIST-Datensets trainiert. [17] Die Testdaten werden mithilfe der Vorhersagen der Kapseln in eine Reihenfolge gebracht. Zur Erklärung dieser Reihenfolge wird ein Decoder verwendet, der die Merkmale des Bildes extrahiert, die zur Erkennung der Klasse führen. Der Decoder wird zudem eingesetzt, um die Merkmale zu ermitteln, die zur Erkennung anderer Klassen beitragen. Die Wahrnehmung innerhalb des Kapselnetzes wird zusätzlich mit künstlich erzeugten Bildern überprüft, die eine Mischung aus zwei Klassen darstellen. Anhand dieser Bilder werden die durch das Kapselnetz erkannten Merkmale ermittelt.

ABSTRACT (ENGLISH)

Search engine rankings are of great significance for their users and providers [1–3]. Thus, they are highly optimized. [4–6] In recent years, the optimization is supported by models of Artificial Intelligence (AI). For AI methods transparency is highly important. Related to AI-generated search engine rankings the term 'transparency' refers to the comprehensibility of the resulting ranking. [7, 8]

Image search and reverse image search could benefit from the application of AI-supported image recognition. Currently, convolutional neural networks (CNN) are applied to recognize images [9–11] The images can be ranked by their relevance for a specific keyword based on the certainty of the CNN's prediction. [12] The knowledge and comprehension of the image areas that impact the certainty of the CNN are the core of explainable AI applications. [12]

Already existing approaches like 'Grad-CAM', 'guided backpropagation' and 'LIME' offer insights into the perception of CNN. [13–15] They show, that CNN tend to use other features than humans for their decisions.

Capsule networks are a specific model architecture that enables the learning of human-understandable features. [16] Therein, special capsules are contained that store the features of objects. These features are, for instance the size or the position of an object, and are consequently comprehensible by humans.

In this thesis, a capsule network is created in Tensorflow and Keras and trained on twelve classes of handwritten letters from the EMNIST dataset. [17] Based on the results for the capsules, a ranking is created for the test images. To explain the ranking in more detail, a decoder is used to examine the image areas that lead to the capsule network's decision. Additionally the decoder is used to restore those image areas that contributed to the recognition of other classes. The perception of the capsule network is additionally tested with artificially created images in which two classes are mixed. The features detected by the capsule network are explained based on these images.

DECLARATION

Ich versichere, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe angefertigt und mich anderer als der in den beigefügten Verzeichnissen angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht.

Der Durchführung einer elektronischen Plagiatsprüfung stimme ich hiermit zu. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht. Ich bin mir bewusst, dass eine unwahre Erklärung rechtliche Folgen hat.

Unna, 12.09.2021 F. Quetscher

Ort, Datum Unterschrift Felizia Quetscher

Contents

List of Figures

List of Tables

1 MOTIVATION

Through the rise of machine learning applications the demand for explainability is increasing. [7, 18] In the report 'Guidelines for Trustworthy AI' the explainability of an AI system is classified as part of its transparency and it consists of two elements: *'the ability to explain [...] the technical processes of an AI system and the related human decisions*'. [8] When the term explainability is used in this thesis, it refers to the technical part of this definition, which is further specified as requirement of an AI system to be '*understood [...] by human beings*' [8]. In this thesis, the term 'explainability' refers to the description and comprehension of the reasons that led to a decision of an AI model.

One large field inside AI is the recognition and classification of objects on images. Currently, the application of convolutional neural networks (CNN) to this task is the state of the art (i.e. [9–11]). Despite their excellent ability for image recognition, classification and segmentation tasks, the decisions of CNN are not directly understandable by the human intuition and thus, their decision-making is difficult to explain.

Multiple approaches provide methods that aim to explain the results and the vision inside CNN by cropping or highlighting important areas of the input image. This is done either by the creation of approximated models [15, 19] or by the additional calculations based on a trained model [13, 14]. Examples for different explanation approaches are provided in figure 1.1.

Figure 1.1: Examples for the results of different explanation approaches. Left to right:
LIME-Approach - original image and result for class 'labrador' [15];
Grad-CAM approach - original image and result for class 'boxer' [13];
Gradient based approach - original image and result (no class provided, image and map slightly cropped) [14]

However, there is no approach yet that leads to a general conclusive solution to explain the vision of CNN. All current results reveal a strong problem with CNN: Despite a correct classification, the features that influenced this decision are neither always human-understandable nor are they always self-explanatory for humans.

Because of the difficult comprehensibility of 'common' CNN, in this thesis a further development of CNN in examined, that promises an improved explainability of classification tasks. The specialized architecture is proposed in [16] and described as 'capsule network'. The core of capsule networks are capsules, that store feature information inside a vector. These features are restored by a decoder and provide insights in the vision of the capsule network.

By [16], the decoded images are presented for the MNIST dataset [20] and it was shown that by changing the squash vector, features such as stroke thickness or angle change based on the squash vector of the capsule network. The property of the capsule network for human-understandable features lays the base for this thesis. Because capsule networks reach comparable results in simple image classification tasks like MNIST [16, 20], they could be an option for a more intuitively understandable model. The goal of this thesis is the examination of this squash vector for its applicability to support the creation of human-understandable network decisions. This ability is tested by the creation of image rankings.

The term 'image ranking' describes the categorization of images based on their affiliation to a specific class. The higher the position of the image in the ranking, the more it is associated with the considered class. The assignment to the specific class is performed by the AI model.

As basis for this topic an overview of existing explanation methods for CNN is provided in chapter 2. Because capsules work significantly different from 'common' CNN, the theoretical background of their algorithm is explained in chapter 3. The capsule network is implemented in Python using Tensorflow and Keras and the model is trained on 12 classes of handwritten letters from the EMINST dataset. [17] The development and the training are described in chapter 4. With the trained model 256 images of each class are predicted and ranked in an order based on the results of the capsules in chapter 5. In section 5.1, the vision of the capsule network is examined by creating decoded images that contain only the information for one class. In section 5.2, further elements recognized by the capsule network are explored. In section 5.3, artificial images containing two mixed letters are created and predicted to analyze the limits of the perception of the capsule network. The results are summarized in chapter 6 and the explanatory approach is evaluated in section 6.1.

2 Explanatory Approaches of Convolutional Neural Networks

Numerous approaches aim to increase the explainability of CNN. This section provides an overview about the basic functionalities of three different methods for CNN explainability: the LIME approach [15], the creation of saliency maps [21] and the Grad-CAM algorithm [13]. These three approaches are selected because they introduce three different methods to explain the results of CNN and all of them are the base for further research.

The LIME ('Local Interpretable Model-agnostic Explanations') approach is a general method to explain single results of AI models. [15] The LIME approach is not limited to any specific model architecture, which is described through the term 'model-agnostic'. The term 'local interpretable' describes that the approach is performed for a single instance, which is either an image or a short part of a text.

The core idea of the LIME approach is the substitution of a multidimensional non-human-understandable model with an easier interpretable but linear model as approximation. For this single model, a new dataset is created that solely contains variations of the single image instance. Each image of the new dataset is modified in different regions, which are either covered with noise or completely removed. By comparing the modified images, the impact of each pixel on each class is examined. The most influential pixels for one class prediction are described as 'super-pixels' and concatenated as key fragments for the decision of the multidimensional model. Examples of the LIME approach are shown in figure 2.1. [15]

Figure 2.1: Example images of the result of the LIME approach [15]. Left to right: Original image, result for class 'labrador', result for class 'acoustic guitar', result for class 'electric guitar'.

While the LIME approach creates a linear approximation of the substituted model, it is further developed in the 'anchor' approach that generates non-linear approximations. [19] The concept of super-pixels is used in this extension as well, but they are not

found by masking areas. Rather, pixels are moved onto different images and the prediction of the neural network is examined. A super-pixel is found if the probability for the class, that belongs to the overlaid pixels, is high. In this case, these super-pixels were decisive for the prediction of the class and are thus described as 'anchors'. Figure 2.2 shows results for an example image presented in [19]. The mid-placed image shows the areas of the left input image calculated as anchors for class 'beagle'. If these anchors are places on another image, for example the right image of figure 2.2, the prediction probability P for the class 'beagle' is still $P \geq 0.9$.

Figure 2.2: Example images of the result of the anchor approach [19]. Left to right: Original image, anchor for class 'beagle', image that generates a probability P for class 'beagle' of $P \geq 0.9$. [19]

The anchor of figure 2.2 shows, that in this case the body of the beagle is irrelevant for the prediction. Instead, a part of the ground seems to impact the result. A similar phenomenon occurs for the prediction of the class 'labrador' in figure 2.1: The lower part of the dog's face is not included in the super-pixels, but a part of the blue shirt is included for this class.

Despite the more or less reasonable location of super-pixels and anchors in figure 2.1 and figure 2.2 as an explanation approach for single images, it does not provide an explanation for the reason of the model's decision. Because they are model-agnostic they can only calculate human-acceptable results but they cannot provide direct insides to the model.

The creation of saliency maps [14] and the Grad-CAM algorithm [13] provide deeper insights into the vision of CNN. These approaches are applied on a specific model. The resulting visualization of these methods is based on highlighting features on the original image, either by a saliency map or by an activation heatmap, which are explained subsequently.

A saliency map displays the prominent regions of an image based on an applied algorithm. In general, there are two different methods to generate saliency maps: The first option is to use a 'deconv' network. [22] A deconv network means the direct copy from the input layer to a specific layer of a CNN. The information of this layer is visualized by

transferring its output backwards through the deconv network. The result is an image with the same dimensions of the input image.

Note, that the term 'deconv' network is chosen to avoid confusions with deconvolutional layers, [23] which are applied in autoencoders. Deconv networks and deconvolutional layers describe different entities.

Because simple copying is not possible for most layers, some modifications are applied in the copied layers. For example the uninvertible maxpooling layer is substituted by a 'switch' layer. This layer uses the positions of the maximum values of its corresponding pooling layer for the unpooling operation. [22]

Figure 2.3: Example saliency maps and their original images from [22] created by the deconv network. The saliency maps in the upper row were created in the fourth layer of the presented model, the saliency maps of the lower row were created in the fifth layer of the model of [22].

The creation of saliency maps with the deconv network provides reasonable insights in specific layers of a CNN. Related to figure 2.3 it is shown again, that the features are perceived in different layers, so that they are independent from each other. This is again related to the problem of the comparison between the vision in CNN and human vision.

The second possibility for creating saliency maps is the usage of backpropagation. [14] It is assumed, that different pixels impact the result of the classification to a specific degree. To infer the impact of each pixel onto the classification, the derivate of the class score with respect to the image input is calculated. A high gradient stands for a high impact of the pixel on the considered class. Pixels with higher impact on the prediction of a class are displayed in stronger colors on the saliency map (figure 2.4). The creation of saliency maps, either by the calculation of the gradient or by the application of a deconv network, provide a reasonable possibility to visualize important areas for a class. As with the aforementioned methods, they work to provide a rough orientation for the important features. They do not directly explain the reason for the CNN's decision.

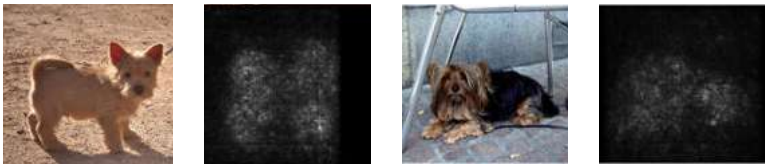


Figure 2.4: Example saliency maps and their original images from [14] created by using the backpropagation approach. Images from [14] slightly cropped.

The last approach to explain the vision in CNN is the creation of activation heatmaps based on the Grad-CAM (Gradient-weighted Class Activation Maps) algorithm. [13] An activation heatmap displays the differently activated regions of an image in specific colors. The Grad-CAM algorithm is an improvement of the CAM algorithm. [24] Within the CAM approach, an activation heatmap is created by the usage of global average pooling (GAP) layers. This layer is an alternative to a dense layer at the end of a CNN. In a dense layer, the information of the last feature maps is flattened and fully connected with the output neurons. With a GAP layer, the average activation of each feature map is calculated. It results in a vector with as many values as there are feature maps. This vector is assumed to contain the confidence for each class and by a softmax layer the probability for each class is received. In comparison to a dense layer, the GAP layer builds a closer connection between feature maps and output neurons. [25]

In the CAM approach the weights of the GAP layer are used to create the activation heat maps. The vector of the GAP layer is multiplied with its corresponding feature map. Because each entry of the GAP vector is a probabiity for a class, the values of feature maps are reinforced based on their impact on the output class. This leads to an estimation of areas that are important for the classification. [24]

The necessity of a GAP layer limits the application possibilities for the CAM algorithm. A larger variety of use cases is covered by the Grad-CAM approach. Within the Grad-CAM, class activation maps are produced, as well, but they do not rely on the GAP layer. Instead, the multiplication factors for the feature maps are calculated differently: Gradients between a change of the feature map's activation and the output value of each class is calculated. The gradients for each feature map are averaged to one value that is subsequently applied onto the feature map. To include only positive values, a ReLU function [26] is applied on the resulting map for each class. The feature maps for one class are concatenated and by the coloration of low and high activated areas the activation heatmaps are created.

Figure 2.5 shows an input image and the corresponding activation heat maps for the classes 'boxer' and 'tiger cat'. The activation heat maps contain a blue to red varying heat map area. According to [13], the blue area represents 'the evidence of the class' and the red area represents a 'high score' of the considered class.

Figure 2.5: Images presented in [13] as input image (left) for the Grad-CAM algorithm, the result for the class 'boxer' (middle) and the result for the class 'tiger cat'. [13]

As the other algorithms described above, the Grad-CAM approach proved a reasonable but coarse orientation for the decision of the CNN. Again, the usage of different criteria of a CNN in comparison to a human become apparent: The most decisive criteria for the decision to class 'tiger cat' are the legs and the hip of the cat because these areas are colored red in figure 2.5. Instead, the head has a similar impact on the decision as the environment that is colored in light blue.

Despite the viable results of the Grad-CAM algorithm, it shows again the deviaton between features used by humans and those used for the CNN's.

3 Theory of Capsule Networks

Capsule networks are special artificial neural networks that are a further development of 'common' CNN. The core of the capsule network's architecture is a structure called 'capsules'. Capsules are a mathematical model proposed in [16] which arranges one set of neurons into multiple subsets (figure 3.1).

The special property that distinguishes capsules from 'common' CNN is an algorithm described as 'dynamic routing'. Thereby, capsules learn to provide their predictions only to those subsequent capsules, that need the preceding prediction for their prediction.

As a consequence, a specialized information transfer is performed within the capsule layer: While in 'common' CNN, information is transferred in scalars, in capsule networks these scalars are concatenated into vectors. The dimension of the vector is equal to the number of neurons contained in a capsules. The capsules visualized in figure 3.1 require a two-dimensional input vector and produce a two-dimensional output vector.

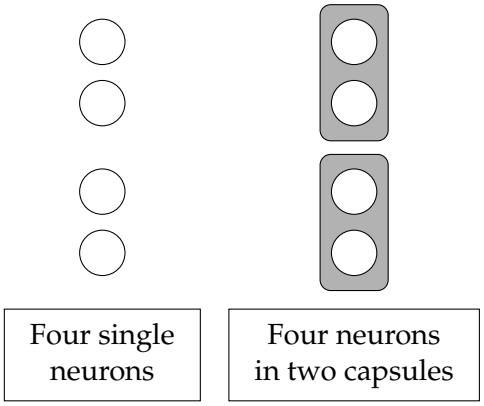


Figure 3.1: Comparison between a model of four single neurons and four neurons arranged in two capsules

The architecture of a simple capsule layer is similar to the architecture of a dense layer: A dense layer consists of an one-dimensional set of input neurons and a subsequent one-dimensional set of output neurons. In contrast to dense layers, capsule layers contain two different connections, the connection by weights in a same way as in a dense layer, and the connection by coupling coefficients. A basic example of a capsule layer is visualized in figure 3.2. It is constructed by two sets of neurons that are arranged into two capsule sets with each capsule containing two neurons. The preceding set of capsules is denoted as 'low-level capsules', because they store low-level features from the preceding layer. The capsules of the subsequent neuron set are denoted as 'high-level capsules'. In

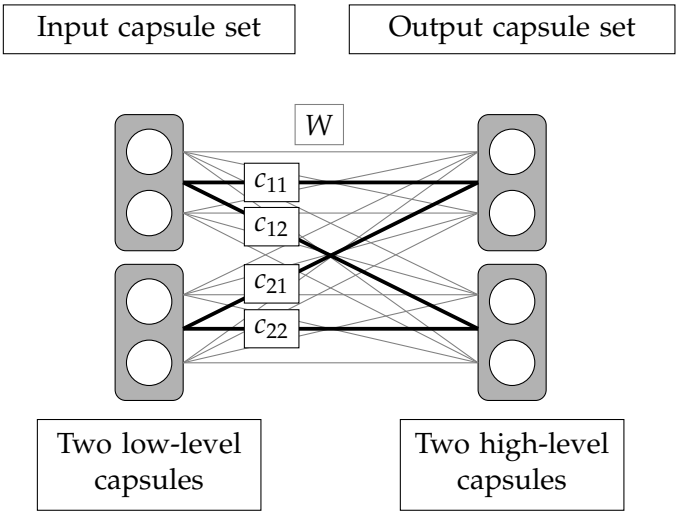


Figure 3.2: Weights and coupling coefficients between two low-level capsules and two high-level capsules

[16], the high-level capsules are used to predict the classes, so that each high-level capsule stores the high-level features of its assigned class.

In the subsequent explanation numerous parameters will be introduced that either relate to the low-level capsules, to the high-level capsules or to both capsules. The letter i is used for low-level capsules and to describe the affiliation of a parameter to the low-level capsules. The letter j is used for high-level capsules and for the affiliation of a parameter to the high-level capsules.

A weight matrix W_{ij} is created between all neurons of each low-level capsule to all neurons of each high-level capsule. Different to a 'common' dense connection, each low-level capsule is additionally connected to each high-level capsule by a connection called coupling coefficient c_{ij}.

While the weight matrix is trained by the application of backpropagation, the coupling coefficients c_{ij} are adjusted in an iterative process called 'dynamic routing'. The idea behind this algorithm is to limit the information transfer of the low-level capsules only to those high-level capsules that use the information for a prediction. This approach was developed as alternative for a maxpooling layer to increase human-understandable vision inside a convolutional neural network. [27] The detailed explanation of the dynamic routing algorithm is presented in section 3.1 based on a minimal example. In section 3.2, the modifications to receive the full capsule network and the resulting parameters are described in section 3.3.

3.1 Exemplary Architecture of a Capsule Network

The architecture of a capsule network depends on the classification task. For an exemplary architecture, a basic example classification task is introduced: The capsule network should be created that distinguishes between both images shown in (figure 3.3).

Image 1 Image 2

Figure 3.3: Two images to be distinguished by a capsule network: Both images consist of two pixels, while one of them is dark and the other one is light.

Both images consist of $[1 \times 2]$ pixels of which the first one is colored light grey and the other pixel is colored dark grey. The difference between both images is the position of the light and the dark pixel. Therefore, the capsule network's output is two-dimensional, either 'image 1' or 'image 2'. Images of two pixels are chosen for this example, because in comparison to one-pixel images, the low-level capsules and high-level capsules are directly assignable to the dark or the light pixel in this example of 2 pixels.
One architecture applicable to this example task is presented in figure 3.4. Each low-level capsule contains two neurons and each high-level capsule contains three neurons. The reason to use an output capsule with three neurons is to better track the transition between vectors belonging to the low-level capsules and vectors belonging to the high-level capsules. Despite this is a minimal capsule network it contains all three elements that are necessary for the architecture of each capsule network:

i At least one convolutional layer with two convolutional filters
ii At least two low-level capsules with each containing two neurons
iii At least one high-level capsule with (at least) two neurons for each output class

The first step for the creation of a capsule layer is the formation of low-level capsules. To create the low-level capsules, based on [16], at least one convolutional layer is necessary. The reason is visualized in figure 3.4 on the following page: By the application of two $[1 \times 1]$ convolutional filters with a step size of 1 pixel and the general absence of zero-padding on the input image, an array with the dimension $[1 \times 2 \times 2]$ is created. This array is described by the term 'Feature Maps'. The low-level capsules are created by concatenating the values along the depth (last axis) of the feature map array. Without any

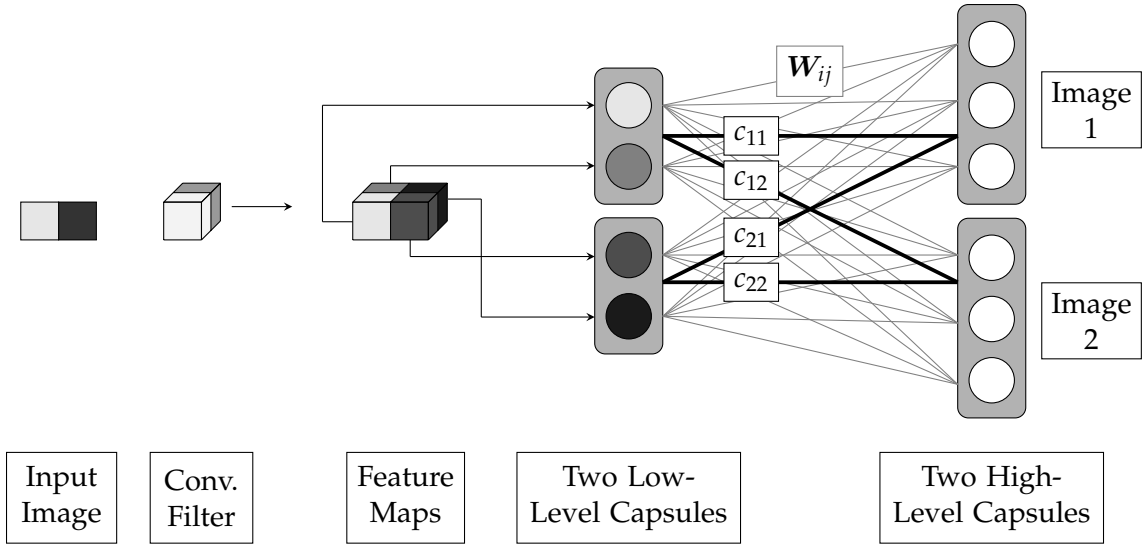


Figure 3.4: A capsule network consisting of two low-level capsules made from two convolutional filters and two output capsules to classify the 2-pixel images of figure 3.3.

further modification, these values are directly transferred into the low-level capsules. An example calculation for the creation of low-level capsules is shown in figure 3.5 which is oriented on figure 3.4. The input image has a light value (0.8) and a dark value (0.2). They are set in this example to generate short, comprehensible values for the feature maps.

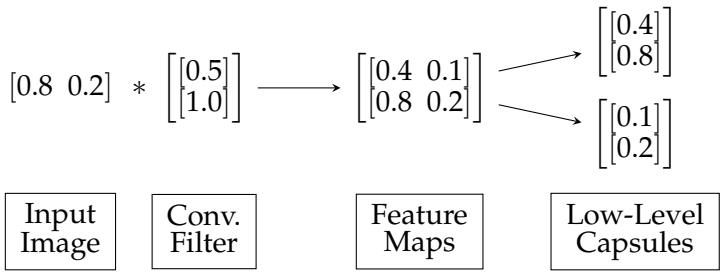


Figure 3.5: Example calculation of two low-level capsules from a two-dimensional input image and two one-dimensional convolutional filters. Calculation corresponds to figure 3.4.

As shown in figure 3.4 and figure 3.5, the creation of low-level capsules is always based on the concatenation of multiple values at the same position along the feature maps. Through this grouping of values from multiple feature maps, features predicted through different convolutional filters are concatenated into one low-level capsule. The low-level capsules store the concatenated values inside a vector whose dimension is equal to the number of neurons in the low-level capsule.

In this simple capsule network, the number of convolutional filters leads directly to the number of neurons per low-level capsule: Both of the convolutional filters generate one value for the vector of the low-level capsule. If the network has considerably more convolutional layers, then not the entire output is stored into one corresponding output capsule. Instead, groups of low-level capsules are formed. This grouping is not necessary in this example and it is discussed in section 3.3 on page 23.
Before the information of the low-level capsules is transferred to the high-level capsules, it is modified by the weight matrix and by the coupling coefficient. This is performed by the dynamic routing which is explained in detail in the subsequent section 3.2. The information in high-level capsules contains the features for the assigned class stored in a vector. As in low-level capsules, the dimension of the vector of a high-level capsule equals the number of neurons of this vector. In this minimal example, the features are stored in a two-dimensional vector. The larger the length of a high-level capsule's vector, the clearer the features are recognized. [16] Therefore, the length of the vector is applied as measurement of the 'certainty' for a class.

3.2 The Dynamic Routing Algorithm

The 'dynamic routing algorithm' is a process that creates specialized connections between low-level capsules and high-level capsules: During the training of the model, for each high-level capsules the low-level capsules are identified, whose predictions are necessary for the prediction of the high-level capsules. To achieve this, the coupling coefficients between low-level capsules and high-level capsules are amplified or mitigated in the training process by a mechanism called 'routing-by-agreement'. Thereby, the orientation of the vector of a low-level capsule is compared to the orientation of the vector of a high-level capsule. If both vectors are oriented similarly or equally, the information of the low-level capsule is assumed to be important for the prediction of the high-level capsule and thus, the connection between both capsules is strengthened.
This learning mechanism inside the capsule layer is based on two parameters: The first parameter is a 'common' weights connection between the neurons. The values of the weight matrix are adjusted through backpropagation. The second parameter is the coupling coefficient that is adjusted through routing-by-agreement. Because the routing-by-agreement mechanism relies on the values of the weight matrix, one training step

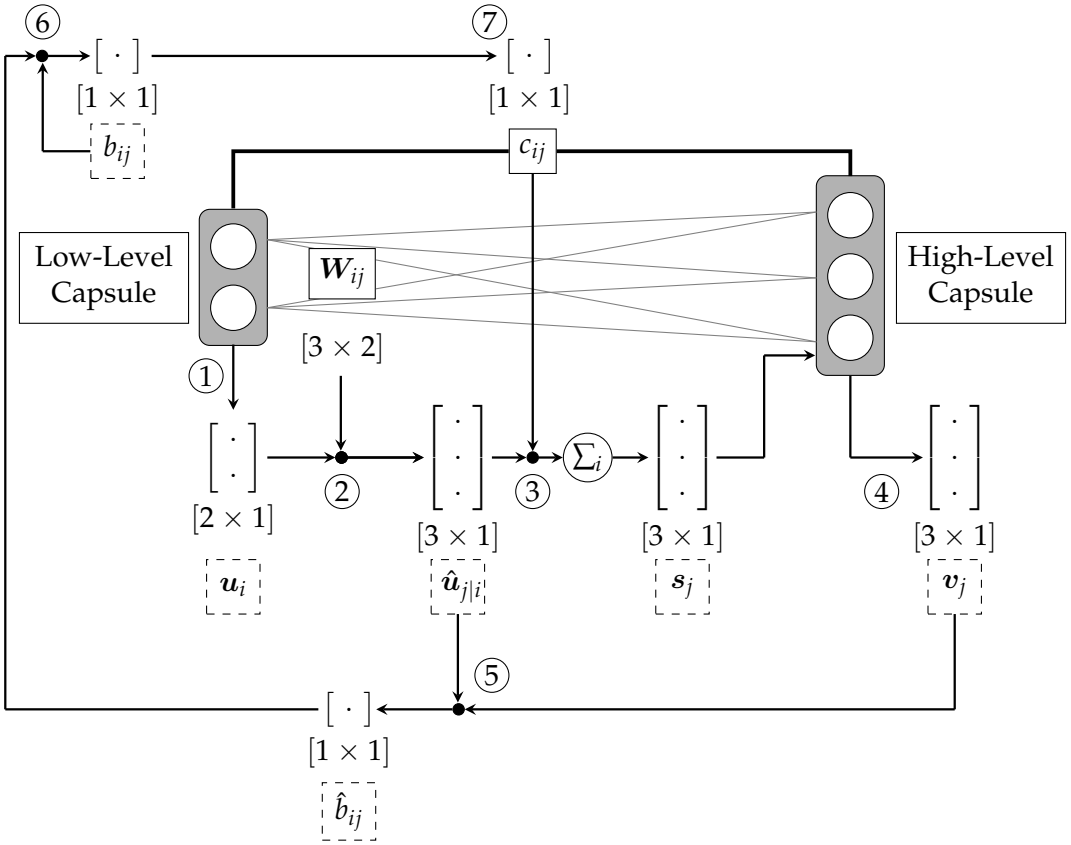


Figure 3.6: Schematic visualization of all tensors involved in a capsule network between a low-level capsule containing two neurons and a high-level capsule containing three neurons.

contains the update of the weight matrix and subsequently, the update of the coupling coefficients.

In figure 3.6, a schematic representation of the dynamic routing algorithm within a capsule layer is displayed. Only a single low-level capsule and a single high-level capsule are shown to increase the clarity of the explanation.

The vectors are represented by braces which visualize the dimension of the vector. Additionally, the dimension is written below the vector, as well as its symbol in dashed rectangles. The symbols are based on the notation used in [16], except for $\hat{\boldsymbol{u}}_{j|k}$ and $\hat{b}_{ij}$, which are not explicitly named in [16]. The numbers $1-7$ display the seven steps of the dynamic routing algorithm. The dynamic-routing approach is explained subsequently based on this steps. In each step, the applied vectors and their meaning for the algorithm are introduced and the corresponding equations are explained.

In **step 1**, the output of the preceding convolutional layer is assigned to the low-level

capsules, as shown in figure 3.4 on page 12. The values are not further modified by the low-level capsule. The resulting vector in the dimension of the low-level capsule is called 'output vector' $\boldsymbol{u}_i$. One output vector exists for each low-level capsule.
In **step 2**, the prediction of the low-level capsule is calculated based on the output vector $\boldsymbol{u}_i$ and on the weight matrix $\boldsymbol{W}_{ij}$ like in a dense layer (equation 3.1).

$$\hat{\boldsymbol{u}}_{j|i} = \boldsymbol{W}_{ij}\,\boldsymbol{u}_i \tag{3.1}$$

By the matrix multiplication of the output vector $\boldsymbol{u}_i$ with the weight matrix $\boldsymbol{W}_{ij}$, the values of $\boldsymbol{u}_i$ are modified and its dimension is adjusted according to the dimension required for the high-level capsule. The resulting vector is called 'prediction vector' $\hat{\boldsymbol{u}}_{j|i}$ because it holds the prediction of the low-level capsule i for the high-level capsule j. In a dense layer, the prediction vector $\hat{\boldsymbol{u}}_{j|i}$ would be used as input to the high-level capsule j. However, in a capsule layer, there is one prediction vector for each connection of a low-level capsule to a high-level capsule. To insert the prediction vectors simultaneously into a high-level capsule, they are initially summarized to one vector. Because each high-level capsule has one assigned class to predict, the output vectors $\boldsymbol{u}_i$ of the low-level capsules vary in their importance for different high-level capsules. This importance is described mathematically in **step 3** by the coupling coefficients that weight the prediction vectors before the summation to the weighted sum $\boldsymbol{s}_j$ (equation 3.2).

$$\boldsymbol{s}_j = \sum_{i=1} c_{ij}\,\hat{\boldsymbol{u}}_{j|i} \tag{3.2}$$

The value of a coupling coefficient is in the interval of 0 and 1, to weight the low-level capsule's prediction vector $\hat{\boldsymbol{u}}_{j|i}$. At the beginning of the training, all coupling coefficients are initialized with equal positive values. In each iteration of the dynamic-routing process, they are adjusted based on the orientation between the vector of the low-level capsule and the vector of the high-level capsule. The update formula of the coupling coefficients is presented in step 7 of the dynamic-routing process.
In the minimal capsule layer of figure 3.6 only a single low-level capsule exists. In a real capsule layer, multiple low-level capsules are connected to one high-level capsule and for each connection a prediction vector $\hat{\boldsymbol{u}}_{j|i}$ exists. The existence of these additional prediction vectors is indicated by the sum symbol. Despite that the coupling coefficients

seem similar to the weights, they are not updated through backpropagation but during the dynamic routing algorithm.

The actual vector input for the high-level capsule j is the weighted sum $\boldsymbol{s}_j$. The values of the weighted sum are impacted either by the presence of features or by a strong connection between the low-level capsule and the high-level capsule. This impact becomes apparent by the concatenation of equation 3.1 (page 15) and equation 3.2 (page 15) to equation 3.3:

$$\boldsymbol{s}_j = \sum_{i=1} c_{ij} \, \boldsymbol{W}_{ij} \, \boldsymbol{u}_i \tag{3.3}$$

The features concatenated in a low-level capsule are stored in $\boldsymbol{u}_i$. If the low-level capsule contained important information for the high-level capsule in the preceding iterations, then c_{ij} and $\boldsymbol{W}_{ij}$ are large resulting in a large weighted sum. On this equation the connection between the dynamic routing algorithm and backpropagation is illustrated: The highest value for the weighted sum is reached only then, if the values of the weight matrix $\boldsymbol{W}_{ij}$ and the values of the coupling coefficient c_{ij} are high.

In **step** 4, the prediction of the high-level capsule is calculated directly from the weighted sum $\boldsymbol{s}_j$ to a vector $\boldsymbol{v}_j$, which has the same dimension as the high-level capsule (equation 3.4).

$$\boldsymbol{v}_j = \frac{||\boldsymbol{s}_j||^2}{(1 + ||\boldsymbol{s}_j||^2)} \, \frac{\boldsymbol{s}_j}{||\boldsymbol{s}_j||} \tag{3.4}$$

Within equation 3.4, the term $||\boldsymbol{s}_j||$ is the euclidean norm of $\boldsymbol{s}_j$, alternatively written as $||\boldsymbol{s}_j||_2$, and is calculated by equation 3.5. [28]

$$||\boldsymbol{s}_j|| = \Big(\sum_{p=1} \boldsymbol{s}_{j,p}^2 \Big)^{\frac{1}{2}} \tag{3.5}$$

The index p represents the entries of the vector $\boldsymbol{s}_j$. If the entries of $\boldsymbol{s}_j$ strongly deviate from zero, the low-level capsules transferred information for the prediction to the high-level capsule in the preceding iterations. In this case, the euclidean norm of $\boldsymbol{s}_j$ is large. If there was no or few information for the high-level capsule in the preceding iterations, the values in $\boldsymbol{s}_j$ are close to zero and thus, the euclidean norm of $\boldsymbol{s}_j$ is small.

The length of $\boldsymbol{s}_j$ directly influences the result of equation 3.4. If the length of $\boldsymbol{s}_j$ is significantly smaller than 1, the first factor of equation 3.4 approaches zero. Through the multiplication of the second term with a value close to 0, the length of the resulting pre-

dicted vector $\boldsymbol{v}_j$ is equally close to 0. However, if the length of $\boldsymbol{s}_j$ is significantly larger than 1, then the first factor of equation 3.4 approaches 1. Then, the result is dominated by the second factor, that normalizes the $\boldsymbol{s}_j$ to a length close to 1.

Consequently, $\boldsymbol{v}_j$ is close to 1 if the features of the class are found in the low-level capsules. If no or only few features are found, the length of $\boldsymbol{v}_j$ is close to 0. This length conversion is denoted in [16] as 'squashing' and the vector $\boldsymbol{v}_j$ of equation 3.4 is called 'squash vector'.

Because the presence of features of a class determines the length of $\boldsymbol{v}_j$ that ranges between 0 and 1, this length is applied as a measurement of the certainty for the assigned class of a high-level capsule.

The squash vector is not only required to create the prediction, it is additionally used for the routing-by-agreement: Because in $\boldsymbol{v}_j$ the predictions of all low-level capsules are summarized into one vector, the individual impact of the single low-level capsules is not verifiable anymore on this point. To examine the impact of each low-level capsule in **step 5**, the squash vector $\boldsymbol{v}_j$ is compared with each prediction vector $\hat{\boldsymbol{u}}_{j|i}$. The orientation of the squash vector $\boldsymbol{v}_j$ of each high-level capsule is compared with the orientation of all its prediction vectors $\hat{\boldsymbol{u}}_{j|i}$ by calculating the dot product between both vectors (equation 3.6).

$$\hat{b}_{ij} = \boldsymbol{v}_j \cdot \hat{\boldsymbol{u}}_{j|i} \tag{3.6}$$

The result $\hat{b}_{ij}$ of the dot product is a scalar that is larger, the closer the orientations of both vectors are to each other. If the vectors $\hat{\boldsymbol{u}}_{j|i}$ and $\boldsymbol{v}_j$ are oriented similar, then $\hat{b}_{ij}$ is large and the prediction of the low-level capsule was decisive for the prediction of the high-level capsule. The similar orientation of the prediction vector $\hat{\boldsymbol{u}}_{j|i}$ and the squash vector $\boldsymbol{v}_j$ is called 'agreement' because the prediction of the low-level capsule supports the prediction of the high-level capsule. The larger the difference in both vector's orientations, the less both vectors agree and the smaller is the impact of the prediction of the low-level capsule onto the prediction of the high-level capsule. The result $\hat{b}_{ij}$ is not only influenced by the orientation of both vectors but additionally by their length. A visualization for the comparison of the orientations and an explanation about their meaning is provided in figure 3.7 by the example of two-dimensional vectors.

The first row of figure 3.7 displays the possible cases if the squash vector $\boldsymbol{v}_j$ and the pre-

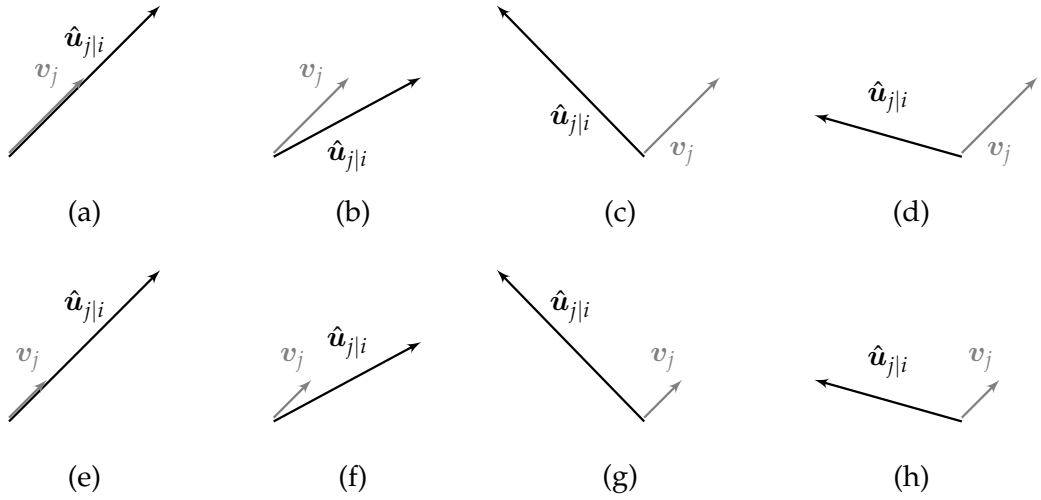


Figure 3.7: Comparison between a long and short squash vector $\boldsymbol{v}_j$ with a long prediction vector $\hat{\boldsymbol{u}}_{j|i}$.

diction vector $\hat{\boldsymbol{u}}_{j|i}$ are long. In (a), both of them are oriented in the same direction which means, they agree completely. This means that the prediction of the low-level capsule is important for the prediction of the high-level capsule. In this situation, $\hat{b}_{ij}$ is large in comparison to all subsequent cases.

The second and third possibility (b) and (c) are different orientations of the prediction vector $\hat{\boldsymbol{u}}_{j|i}$ and the squash vector $\boldsymbol{v}_j$. In (b), the case of slightly different oriented vectors is shown. This case leads to a smaller value of $\hat{b}_{ij}$ compared to case (a). Accordingly, the impact of the prediction of the low-level capsule to the prediction of the high-level capsule is smaller than in case (a), but still significant. In case (c), both vectors are orthogonal to each other. Related to equation 3.6, the resulting $\hat{b}_{ij}$ is 0. In this case, the prediction of the previous low-level capsule has no impact on the prediction of the high-level capsule. This high-level capsule used the features of one or multiple other low-level capsules for its prediction. In this case, the prediction of the low-level capsule was scaled down either by the weight matrix or by the coupling coefficient (compare equation 3.3). In case (d), the prediction of $\hat{\boldsymbol{u}}_{j|i}$ is similarly unrelated to the prediction of $\boldsymbol{v}_j$, but a negative routing logit is derived as result to weaken the connection between the capsules.

The remaining cases (e-g) show the situation with a large prediction vector $\hat{\boldsymbol{u}}_{j|i}$ and a small squash vector $\boldsymbol{v}_j$. In case (e), the prediction of the low-level capsule is necessary for the prediction of the high-level capsule, but it was scaled down by the coupling coefficient. This is possible during the training, if the coupling coefficients are not yet adjusted

correctly and thus decrease the value of important prediction vectors (compare equation 3.2). Case (e), (f) and (g) display a large prediction vector $\hat{\boldsymbol{u}}_{j|i}$ but with small or no impact on capsule j. In this case, the prediction vector was not relevant to the prediction of high-level capsule.

The parameter that results from the comparison of the orientations is the update routing logit $\hat{b}_{ij}$. In **step 6** it is applied to update the connection strength between the corresponding low-level capsule and the high-level capsule for the next routing iteration. Despite that each connection between a low-level capsule and a high-level capsule is described by the coupling coefficients, the routing logits are the direct precursors of the coupling coefficients. At the beginning of the training, each routing logit is initialized to zero. With each routing iteration, the current routing logit $\tilde{b}_{ij}$ is updated with the update routing logit $\hat{b}_{ij}$ (equation 3.7).

$$b_{ij} = \tilde{b}_{ij} + \hat{b}_{ij} \tag{3.7}$$

This equation demonstrates, that all routing logits are modified in each routing iteration. The magnitude of the update depends on the agreement of the low-level capsule and the high-level capsule in step 6: The update of the routing logit is large if both capsules agreed. Otherwise, the routing logits are updated to a smaller degree.

In **step 7**, the coupling coefficients c_{ij} are calculated for the next routing iteration by the softmax-like equation 3.8 based on the new routing logits.

$$c_{ij} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ik})} \tag{3.8}$$

In this equation, the index k represents the indices of all output capsules, while j represents the considered output capsule. The considered coupling coefficient c_{ij} does not only result from the importance of the low-level capsule i for the considered high-level capsule j but also from the low-level capsule's importance to all other high-level capsules k. The sum of all coupling coefficients of a low-level capsule is always 1.

At the beginning of the training, all routing logits were initialized to zero, so that the size of the coupling coefficient depends on the number of capsules.

During the training progress, a large coupling coefficient close to 1 is generated if one specific high-level capsule needs the result of the low-level capsule significantly more than all other high-level capsules. In this case, the other coupling coefficients converge to

0. If multiple high-level capsules have a comparable high necessity for the prediction of the low-level capsule, the value of the coupling coefficients for each of these connections converges to $1/q$, with q being the number of high-level capsules that need the prediction. A similar phenomenon occurs if all connections are equally unimportant. In this case, all routing logits would get updated by the same small degree and result in equal coupling coefficients.

The core of the described dynamic routing algorithm is the squash vector $\boldsymbol{v}_j$. To encourage the capsule network to create long squash vectors for a present class and short squash vectors of a non-present class, a special loss function is applied (equation 3.9). This loss function is calculated individually for each high-level capsule j and covers both cases of a short and of a long squash vector.

$$L_j = T_j \max(0, m^+ - ||\boldsymbol{v}_j||)^2 + \lambda\ (1 - T_j) \max(0, ||\boldsymbol{v}_j|| - m^-)^2 \tag{3.9}$$

The parameter T_j is different depending on the class present in the input image: If the class assigned to the high-level capsule j is present in the image, T_j is 1 and the first addend of the equation counts for the loss calculation. Equivalently, T_j is set to 0, if the class of the capsule is not present in the image. Thereby, the second addend of equation 3.9 counts for the loss calculation. By the parameters m^+ and m^- the targeted length of $\boldsymbol{v}_j$ is set. In [16], the parameters are set to $m^+ = 0.9$ and $m^- = 0.1$. As a result, the loss for the present class drops to zero, if the correct squash vector $\boldsymbol{v}_j$ has a length larger than m^+. For a non-present class, the loss drops to 0, if the squash vector has a length shorter than m^-. Through m^+ and m^- a margin is created for the length of the squash vectors. Hence, in equation 3.9, this loss function is called 'margin-loss'.

To strengthen the impact of a correct prediction, the parameter λ is applied as downscaling factor for the loss of capsules, whose class is not present in the image. Thereby, the importance of a long squash vector $\boldsymbol{v}_j$ for the correct class is increased in comparison to a short squash vector $\boldsymbol{v}_j$ for an incorrect class. Equation 3.9 is basically a modified ReLU function [26] , because all values under or over a specific value, respectively, result in the value of 0.

This loss function (equation 3.9) has a peculiarity: Based on the structure of the network, exactly one squash vector for each class exists. This results in exactly one prediction per class. Even if two instances of one class are present in an image, the capsule network

Table 3.1: Vectors for one high-level capsule in the example capsule network of figure 3.4

Step	Eq.	Input				Result	
1	-	$[0.40], [0.80]$				$u_1 =$	$\begin{bmatrix}0.40\\0.80\end{bmatrix}$
		$[0.10], [0.20]$				$u_2 =$	$\begin{bmatrix}0.10\\0.20\end{bmatrix}$
2	3.1	$u_1 = \begin{bmatrix}0.40\\0.80\end{bmatrix}$	$W_{11} = \begin{bmatrix}1.0\ 1.0\\1.0\ 1.0\\1.0\ 1.0\end{bmatrix}$			$\hat{u}_{1\|1} =$	$\begin{bmatrix}1.20\\1.20\\1.20\end{bmatrix}$
		$u_2 = \begin{bmatrix}0.10\\0.20\end{bmatrix}$	$W_{21} = \begin{bmatrix}1.0\ 1.0\\1.0\ 1.0\\1.0\ 1.0\end{bmatrix}$			$\hat{u}_{1\|2} =$	$\begin{bmatrix}0.30\\0.30\\0.30\end{bmatrix}$
3	3.2	$\hat{u}_{1\|1} = \begin{bmatrix}1.20\\1.20\\1.20\end{bmatrix}$	$\hat{u}_{1\|2} = \begin{bmatrix}0.30\\0.30\\0.30\end{bmatrix}$	$c_{11} = 0.5$	$c_{21} = 0.5$	$s_1 =$	$\begin{bmatrix}0.75\\0.75\\0.75\end{bmatrix}$
4	3.4	$s_1 = \begin{bmatrix}0.75\\0.75\\0.75\end{bmatrix}$				$v_1 =$	$\begin{bmatrix}0.36\\0.36\\0.36\end{bmatrix}$
5	3.6	$\hat{u}_{1\|1} = \begin{bmatrix}1.20\\1.20\\1.20\end{bmatrix}$	$v_1 = \begin{bmatrix}0.36\\0.36\\0.36\end{bmatrix}$			$\hat{b}_{11} =$	1.30
		$\hat{u}_{1\|2} = \begin{bmatrix}0.30\\0.30\\0.30\end{bmatrix}$	$v_1 = \begin{bmatrix}0.36\\0.36\\0.36\end{bmatrix}$			$\hat{b}_{12} =$	0.32
6	3.7	$\hat{b}_{11} = 1.30$	$\tilde{b}_{11} = 0$			$b_{11} =$	1.30
		$\hat{b}_{12} = 0.32$	$\tilde{b}_{12} = 0$			$b_{12} =$	0.32
7	3.8	$b_{11} = 1.30$	$b_{12} = 0.32$			$c_{11} =$	0.73
		$b_{11} = 1.30$	$b_{12} = 0.32$			$c_{12} =$	0.27

will only predict the existence of the class, but not the number of instances. Although different low-level capsule could detect different instances because they are in in different areas, all their information would be transferred to the same high-level capsule. This high-level capsule only has the possibility to recognize the transferred features, but it cannot distinguish if they originated from two different instances.

The calculation of the loss of the complete capsule network is performed by summation of the losses L_j (equation 3.10).

$$L = \sum_j L_j \tag{3.10}$$

Through the loss function for each class, a relation between the length of the squash vector $\boldsymbol{v}_j$ and the weight matrix $\boldsymbol{W}_{ij}$ is created. By minimizing the loss through backpropagation, only the weight matrix $\boldsymbol{W}_{ij}$, but not the coupling coefficients c_{ij} are modified. However, by the usage of $\boldsymbol{v}_j$ in the loss function, an indirect relation between the coupling coefficients and the weights is created.

To illustrate this relation, the vectors of the dynamic routing between two low-level capsules and the upper high-level capsule of figure 3.4 are calculated. The calculation is based on the example visualized in figure 3.2 on page 10 and the results are shown in table 3.1 sorted for the steps previously described. The calculation covers the first iteration of the dynamic routing approach. Therefore, the 'preceding' routing logits have the value 0, as noted in step 6. In the first iteration, their value is a direct consequence of the number of high-level capsulesİn this example both coupling coefficients have a value of 0.5.

In the first step, the input to the low-level capsules are the values of the feature maps, whose calculation is illustrated previously in figure 3.4. For the second step, the values of weight matrix for the calculation of the prediction vector $\hat{\boldsymbol{u}}_{j|i}$ are exemplary set to 1. The result of this exemplary calculation are the coupling coefficients. The coupling coefficient c_{11} from the first low-level capsule to the first high-level capsule is 0.73 and the coupling coefficient c_{12} of the second low-level capsule to the first low-level capsule is 0.27. These values demonstrate that the coupling is oriented towards the larger prediction vector.

The weight matrix $\boldsymbol{W}_{ij}$ is a part of the calculation for the prediction vector. For the discussed example, both weights matrices were set equally. However, in normal cases, the values of the weight matrices contain different values. To demonstrate the effect of two different matrices, the example of table 3.1 is modified by different values in the weight matrix. The results are illustrated in table 3.2. Therein, the weight matrix for upper high-level capsule for image 1 has different values.

In table 3.2, the consequence of a larger weight matrix is demonstrated: In this scenario, the coupling coefficient c_{11} is significantly larger in comparison to the coupling coefficient in table 3.1.

This shows, that the weight matrix and the coupling coefficient are related: The weights impact the squash vector through the prediction vector and the weighted sum (equation 3.3). If the values of the weight matrix are increased by backpropagation, the weighted sum and the squash vector become larger in comparison to the previous iteration. A larger squash vector leads to a larger routing update and consecutively, to a larger coupling coefficient for the subsequent routing iteration (equation 3.6). A larger coupling coefficient and a larger weight matrix increase both the importance of the low-level capsule to the considered high-level capsule (equation 3.1). [16]

Table 3.2: Vectors for one high-level capsule in the minimal example capsule network of figure 3.4 with different weight matrices

Step	Eq.	Input				Result	
1	-	$[0.40], [0.80]$				$u_1 =$	$\begin{bmatrix}0.40\\0.80\end{bmatrix}$
		$[0.10], [0.20]$				$u_2 =$	$\begin{bmatrix}0.10\\0.20\end{bmatrix}$
2	3.1	$u_1 = \begin{bmatrix}0.40\\0.80\end{bmatrix}$	$W_{11} = \begin{bmatrix}2.0 & 2.0\\2.0 & 2.0\\2.0 & 2.0\end{bmatrix}$			$\hat{u}_{1\|1} =$	$\begin{bmatrix}2.40\\2.40\\2.40\end{bmatrix}$
		$u_2 = \begin{bmatrix}0.10\\0.20\end{bmatrix}$	$W_{21} = \begin{bmatrix}1.0 & 1.0\\1.0 & 1.0\\1.0 & 1.0\end{bmatrix}$			$\hat{u}_{1\|2} =$	$\begin{bmatrix}0.30\\0.30\\0.30\end{bmatrix}$
3	3.2	$\hat{u}_{1\|1} = \begin{bmatrix}2.40\\2.40\\2.40\end{bmatrix}$	$\hat{u}_{1\|2} = \begin{bmatrix}0.30\\0.30\\0.30\end{bmatrix}$	$c_{11} = 0.50$	$c_{21} = 0.50$	$s_1 =$	$\begin{bmatrix}1.35\\1.35\\1.35\end{bmatrix}$
4	3.4	$s_1 = \begin{bmatrix}1.35\\1.35\\1.35\end{bmatrix}$				$v_1 =$	$\begin{bmatrix}0.48\\0.48\\0.48\end{bmatrix}$
5	3.6	$\hat{u}_{1\|1} = \begin{bmatrix}2.40\\2.40\\2.40\end{bmatrix}$	$v_1 = \begin{bmatrix}0.48\\0.48\\0.48\end{bmatrix}$			$\hat{b}_{11} =$	3.46
		$\hat{u}_{1\|2} = \begin{bmatrix}0.30\\0.30\\0.30\end{bmatrix}$	$v_1 = \begin{bmatrix}0.48\\0.48\\0.48\end{bmatrix}$			$\hat{b}_{12} =$	0.43
6	3.7	$\hat{b}_{11} = 3.46$	$\tilde{b}_{11} = 0$			$b_{11} =$	3.46
		$\hat{b}_{12} = 0.43$	$\tilde{b}_{12} = 0$			$b_{12} =$	0.43
7	3.8	$b_{11} = 3.46$	$b_{12} = 0.43$			$c_{11} =$	0.95
		$b_{11} = 3.46$	$b_{12} = 0.43$			$c_{12} =$	0.05

3.3 Extension of the Example for Large-Scale Capsule Networks

The presented minimal example includes all elements necessary for the construction of a capsule network. However, when a large number of convolutional layers is applied to increase the capsule network, the size of the low-level capsules becomes a new adjustable parameter: Opposite to the example shown in figure 3.4, in a large-scale capsule network, the size of the low-level capsules is not necessarily derived from the number of feature maps. Instead, a parameter for the number of neurons per low-level capsule is used. Its meaning is described by an enlarged minimal example shown in figure 3.8 with a number of feature maps $n_m = 4$. Based on the previously presented method for the creation of low-level capsules, the number of neurons in one low-level capsule would be 4. In this case, 256 convolutional filters would lead to 256 neurons in one low-level capsule. To avoid large-size low-level capsules, only a specific fixed number of values along the feature maps are concatenated into one low-level capsules. This results in the creation of

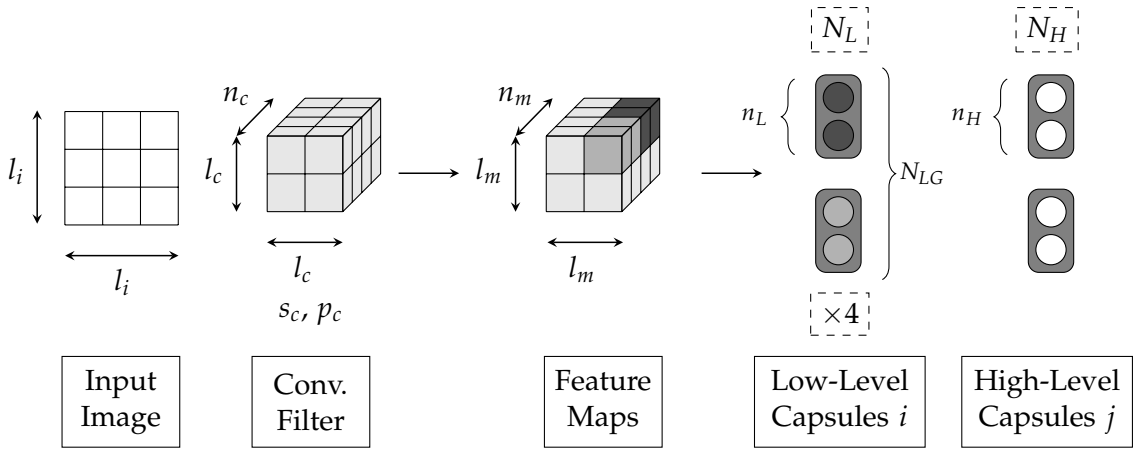


Figure 3.8: Exemplary architecture of a capsule network that contains the formation of capsule groups. The architecture is based on a greyscale input image.

multiple low-level capsules from one position in the feature maps.

In figure 3.8, the number of neurons per low-level capsule is set to 2. All 4 feature maps are concatenated in 2 groups of 2 neurons which are visualized for one position in the feature maps by the light and dark colored neurons. Each color represents a different group. This grouping is performed for each position of the feature map, leading to 4×2 low-level capsules. The neurons of one position in one feature map group are concatenated into one low-level capsule. As a consequence, multiple low-level capsules along the axis of the feature map are possible. Furthermore, this means that the low-level capsules can specialize on features of different classes that are located at the same position. The number of neurons per low-level capsule is equal to the number of features storable in the low-level capsule. Multiple smaller low-level capsules support the detection of more features because more low-level capsules are available. Few large low-level capsules support the detection of fewer features but the features stored inside a low-level capsule may vary stronger.

For the construction of the capsule network, the number of neurons per low-level capsule must fit to the number of feature maps. This connection is expressed by equation 3.11, in which n_L is the number of neurons per low-level capsule and n_m is the number of feature maps. The parameter N_{LG} is the number of capsule groups that are created along the axis of the feature maps.

$$n_m = n_L \cdot N_{LG} \tag{3.11}$$

Table 3.3: Overview for construction-related parameters of a capsule network in relation to the specific example in figure 3.8. Parameters that are not directly adjusted but influenced by other parameters are marked with an asterisk.

Parameter	Symbol	Value in Figure 3.8 / Pixel
Input Image		
Image side length	l_i	3
Convolution Filter		
Conv. filter side length	l_c	2
Number of conv. filter	n_c	4
Stride	s_c	1
Zero-Padding	p_c	0
Feature Maps		
Feature Map Side Length*	l_m	2
Number of Feature Maps*	n_m	4
Low-level capsules		
Number of neurons per low-level capsule	n_L	2
Number of low-level capsule groups*	N_{LG}	2
Number of low-level capsules*	N_L	8
High-level capsules		
Number of neurons per high-level capsule	n_H	2
Number of high-level capsule*	N_H	2

With the possibility for low-level capsule groups, any large-scale capsule network related to [16] is constructible. All construction-related parameters are summarized in table 3.3 for the capsule network shown in figure 3.8. In the construction of the capsule network, the size of the input image l_i and the settings for the convolutional filters are adjustable in the general rules for convolutional arithmetic. [29]

The side length and number of the feature maps are not directly adjustable: The side length l_m is dependent of the setting of the preceding convolutional layer. It is calculable by the size l_i of the quadratic input image, the size of the quadratic convolutional filter l_c, their stride s_c and their padding p_c (equation 3.12). [29]

$$l_m = \left\lfloor \frac{l_i - l_c + 2p_c}{s_c} \right\rfloor + 1 \tag{3.12}$$

The number of feature maps n_m is a direct result of the number of the previously applied convolutional filters n_c (equation 3.13).

$$n_c = n_m \tag{3.13}$$

The number of neurons n_L per low-level capsule is freely selectable. It leads with the number of the feature maps n_m to the number N_{LG} of low-level capsule groups based on equation 3.11. The number of low-level capsules results from the side length l_m of the feature maps, the number of feature maps n_m and the number N_{LG} of low-level capsule groups (equation 3.14).

$$N_L = l_m^2 \cdot N_{LG} \tag{3.14}$$

Instead of setting the number n_L of neurons per low-level capsule, the number N_{LG} of low-level capsule groups or the number N_L of low-level capsules can be selected and the remaining parameters are calculated based on equation 3.11 and equation 3.14.

The number N_H of high-level capsules is equal to the number of classes inside the provided dataset. The number n_H of neurons per high-level capsule is freely selectable. This number is the dimension of the squash vector $\boldsymbol{v}_j$ and it impacts the number of features storable in one high-level capsule. Equally to the number of neurons in the low-level capsules, the number of neurons in the high-level capsules impacts the storable features in the squash vector: With a higher number of neurons, more features may be stored, because the information of more low-level capsules is representable in the high-level capsules. At the same time, the training difficulty rises, because more weights per capsule must be optimized. With a lower number of neurons in the high-level capsules the storable features and the number of training parameters in one high-level capsule are reduced.

Another additional factor emerges for the training of large-scale capsule networks: While the dynamic routing was introduced to take place directly after the update of the weight matrix, this is not the only option for the frequency that the dynamic routing can be executed. Because it impacts size of the coupling coefficients based on a given weight matrix, the 'common' backpropagtion-based training of the weights must be balanced to the coupling coefficients' adjustment through the dynamic routing algorithm. The weight matrix gets updated after a batch of images passed the capsule network. After this update, new coupling coefficients are calculated. However, based on these coupling coefficients the dynamic routing algorithm inside the capsule layer can be executed another time which results in different coupling coefficients. Consequently, the number of routings per training step is an adjustable factor, to be set in large-scale capsule networks. In [16] the

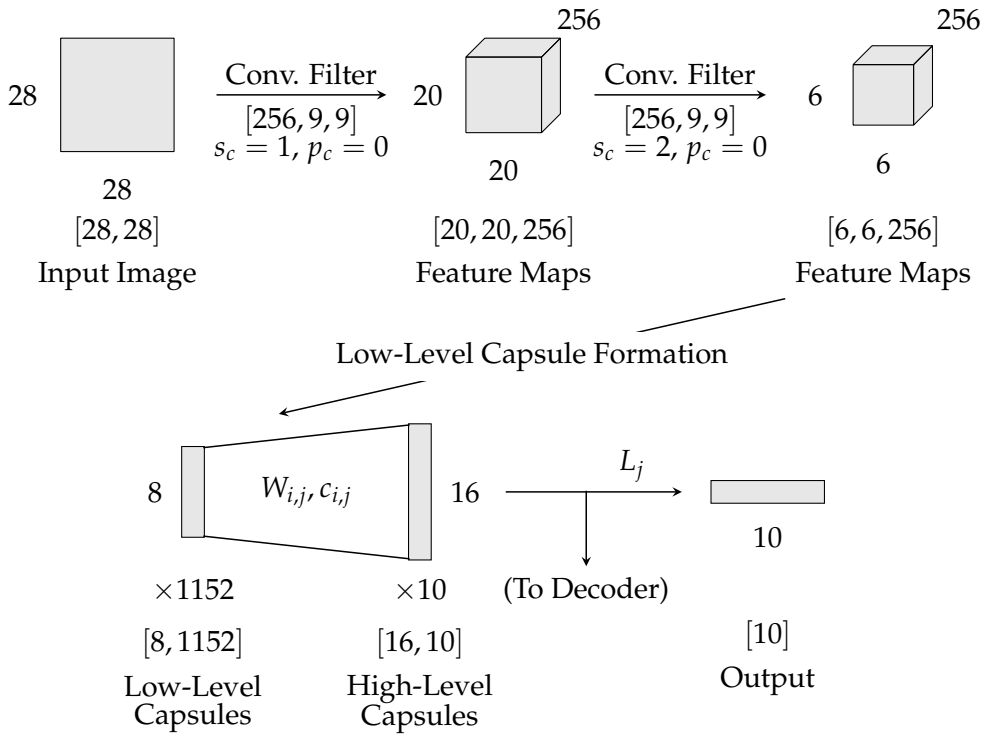


Figure 3.9: Architecture of a capsule network proposed in [16] with a greyscale image input side length of 28 pixels. Because of the usage of this capsule network for the MNIST dataset, the number of output capsules is 10.

recommended option to train on the MNIST dataset is a routing number of $r = 3$. The architecture of the capsule network proposed by [16] is visualized in figure 3.9. Different from a basic capsule network, the architecture contains two subsequent convolutional layers at the beginning and additionally, a decoder at the end by which the perceived images are reconstructed. Because of this decoder, the complete structure of [16] containing the capsule network and the decoder is subsequently denoted as 'capsule-decoder-network'.

The parameters of the capsule-decoder-network are summarized in table 3.4. The first two layers are convolutional layers containing 256 convolutional filters with a side length of $l_1 = l_2 = 9$. In the first convolution layer, a stride of $s_c = 1$ is applied and in the second convolution layer a stride of $s_c = 2$. After both convolutional layers 256 feature maps with a side length of $l_m = 6$ are created. The number of neurons per low-level capsule is set to 8 and, therefore, the low-level capsules are arranged into 32 low-level capsule groups.

The model is created for the training of the MNIST dataset so that the number of high-level capsules is set to 10. Each of the high-level capsules contains 16 neurons, creating

Table 3.4: Parameters of the capsule-decoder-network's architecture proposed in [16]

Parameter name	Parameter Symbol	Dimension
Input Image		
Side length	l_i	28
Convolutional Layer 1		
Kernel Number	N_1	256
Kernel Size	k_1	9
Stride	s_1	1
Padding	p_1	0
Convolutional Layer 2		
Kernel Number	N_2	256
Kernel Size	k_2	9
Stride	s_2	2
Padding	p_2	0
Feature Maps		
Feature Map Side Length	l_m	6
Number of Feature Maps	n_m	256
Low-level capsules		
Number of neurons per low-level capsule	n_L	8
Number of low-level capsules	N_L	1152
Number of low-level capsule groups	N_{LG}	8
High-level capsules		
Number of neurons per high-level capsule	n_H	16
Number of high-level capsule	N_H	12
Capsule Network Output Layer		
Number of output neurons	n_{ON}	12
Decoder Network		
Number of dense neurons (Layer 1)	n_{D1N}	512
Number of dense neurons (Layer 2)	n_{D2N}	1024
Number of output neurons	n_{DON}	784

a 16-dimensional squash vector $\boldsymbol{v}_j$. The squash vector of the high-level capsules is used as input for the decoder. Therefore the array with the dimension $[10 \times 16]$ is flattened and used as input for the subsequent dense layer. A detailed visualization of the decoder is provided in figure 3.10 on the facing page. The decoder consists of three sequential dense layers with 512, 1024 and 784 neurons, respectively. The first two layers use the ReLU function [26] as activation and the last layer uses the sigmoid function [30]. The number of neurons in the last layer is defined by the square of the size l_i of the input image. The output of this layer is subsequently reshaped into an two-dimensional image with the size of $[28 \times 28]$. Because the decoder is attached to the capsule network, the training of both models is connected. While the capsule network is trained by providing

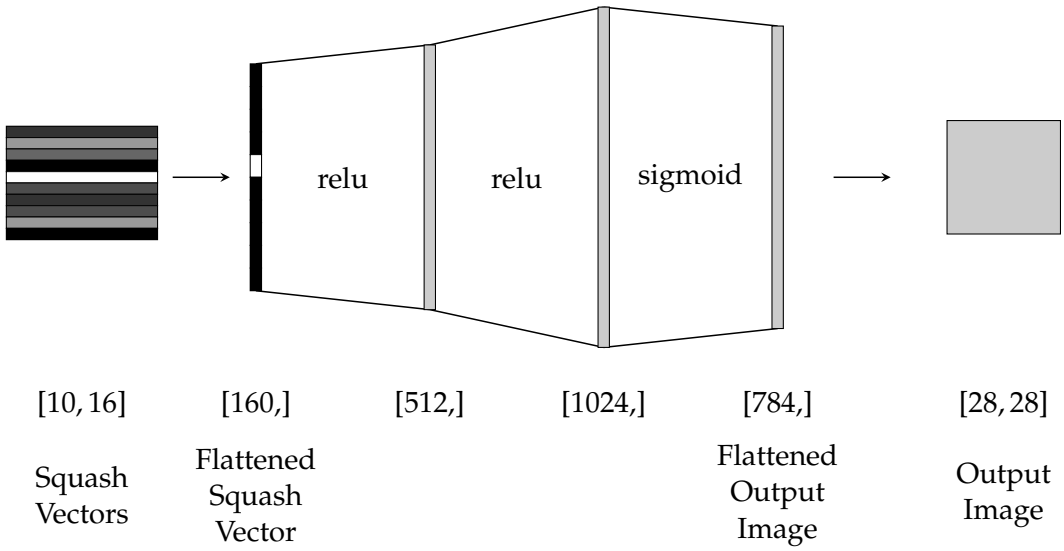


Figure 3.10: Visualization of the decoder using the squash vectors of the capsule network

the class labels, the decoder part is trained by providing the input images. The capsule network is trained based on the margin-loss function (equation 3.9). For the complete capsule-decoder-network, the euclidean norm is calculated between the prediction $x_{pred,i}$ and the true value $x_{true,i}$ of all predicted pixels p (equation 3.15).

$$L_D = \left(\sum_{i=1}^{p} (x_{pred,i} - x_{true,i})^2 \right)^{\frac{1}{2}} \tag{3.15}$$

For the training of the capsule-decoder-network, all squash vectors except for the one of the present class are set to zero in a process called 'masking'. Therefore, the decoder learns a general representation of the class at a specific position of the decoder. However, during testing no masking is executed. The reason for that lays in the learning ability of the capsule network: A properly trained capsule network leads to small values in those squash vectors whose class is non present in the image, making a masking unnecessary. The decoder is not only responsible for reconstructing the images, rather it provides a 'motivation' for the capsule network to learn the class representations inside the capsules. To ensure the representation learning of the capsules, the loss of the capsule network L (equation 3.9 on page 20) and the loss of the capsule-decoder-network L_D (equation 3.15) are weighted and added to one total loss L_total (equation 3.16).

$$L_{total} = L + d \cdot L_D \tag{3.16}$$

In equation 3.16, d is the weighting factor for the loss of the capsule-decoder-network in relation to the loss of the capsule network. The weighting factors are additional hyper-parameters to control the output of the capsule-decoder-network.

4 Development of a Capsule-Decoder-Network in Tensorflow

To investigate capsule networks for their applicability to image rankings, a capsule network is created in Python using Tensorflow and Keras functions. In this chapter, the construction of the capsule network and the training process is described.
The capsule-decoder-network is trained on 12 classes of the balanced EMNIST letters dataset, subsequently referred to as 'reduced EMNIST dataset'. [17] This dataset contains white handwritten letters and digits on a black background. Examples of the 12 used classes of the EMNIST dataset are shown in figure 4.1. The classes were chosen to cover a large variety in the structure of the letters. Though some classes of the balanced EMNIST dataset contain lower and upper case letters, for the training of the capsule-decoder-network only capital letters are used.

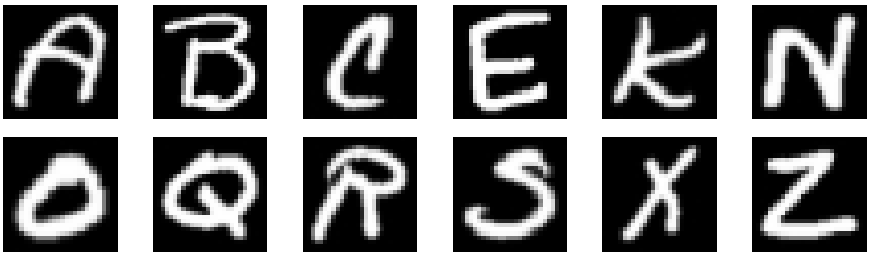


Figure 4.1: Examples of training images of the EMNIST dataset [17]

Generally, for the creation of a model in Tensorflow all calculations must be known including the resulting dimensions of the corresponding tensors. In the practical application each tensor is composed of more dimensions compared to the theoretical description of the capsule network. This results from the processing of tensors by Tensorflow. In section 4.1 on the following page, the dimensions of the tensors involved in the capsule-decoder-network and in the capsule layer are described and visualized. Capsule layers are not yet available in Tensorflow or Keras, requiring them to be programmed individually. The code focusses on the development of the dynamic routing between the low-level capsules and the high-level capsules. In comparison to a common dense layer, multiple additional parameters are necessary for the creation and execution of the capsule layer.
In section 4.2 the training of the created model onto the reduced EMNIST dataset and the training results are described.

4.1 Implementation of the Capsule-Decoder Network in Tensorflow

A large part of the capsule network and of the decoder are implemented with predefined Tensorflow functions. However, custom functions are applied in the capsule network for all capsule-related calculations and in the decoder for the masking of the squash vectors. The dimensions of the tensors are derived by the architecture of the capsule network in [16] and are summarized in table 3.4 on page 28. Deviating from [16], in this implementation 12 low-level capsules are used because the dataset contains 12 classes. Additionally, the number of neurons in the first and second dense layer of the decoder are set to 2048 and 4096 neurons, respectively. A larger number of neurons is assumed useful for the decoder as handwritten letters contain more features compared to handwritten digits.

In figure 4.2 on the facing page, an overview of the layers of the capsule-decoder-network is provided including the dimensions for all tensors. The layers are visualized by grey-colored rectangles and connected by arrows. These arrows demonstrate the process of tensors inside the capsule-decoder-network. The dimension of the tensors are written in square brackets next to each arrow to show the output of one layer as input to the subsequent layer(s). The symbols are equivalent to those introduced in table 3.4 except for b_i which is the symbol for the batch size.

When a model is training or predicting with Keras' '.fit()', '.evaluate()' or '.predict()' method, the batch size is stored as the first dimension of the tensor. Therefore, the batch size is be included in the calculation of the capsule-decoder-network model. Because it is independent from the tensor dimensions, its value is stated as -1.

By figure 4.2 the difference between the theoretical calculation of the tensors and the actual implementation of the model is apparent: In a model, not only one vector but rather a multidimensional tensor is processed. Not only the batch size must be included in each calculation, but also the shapes of the tensors must be adapted to the functions provided by Tensorflow and Keras.

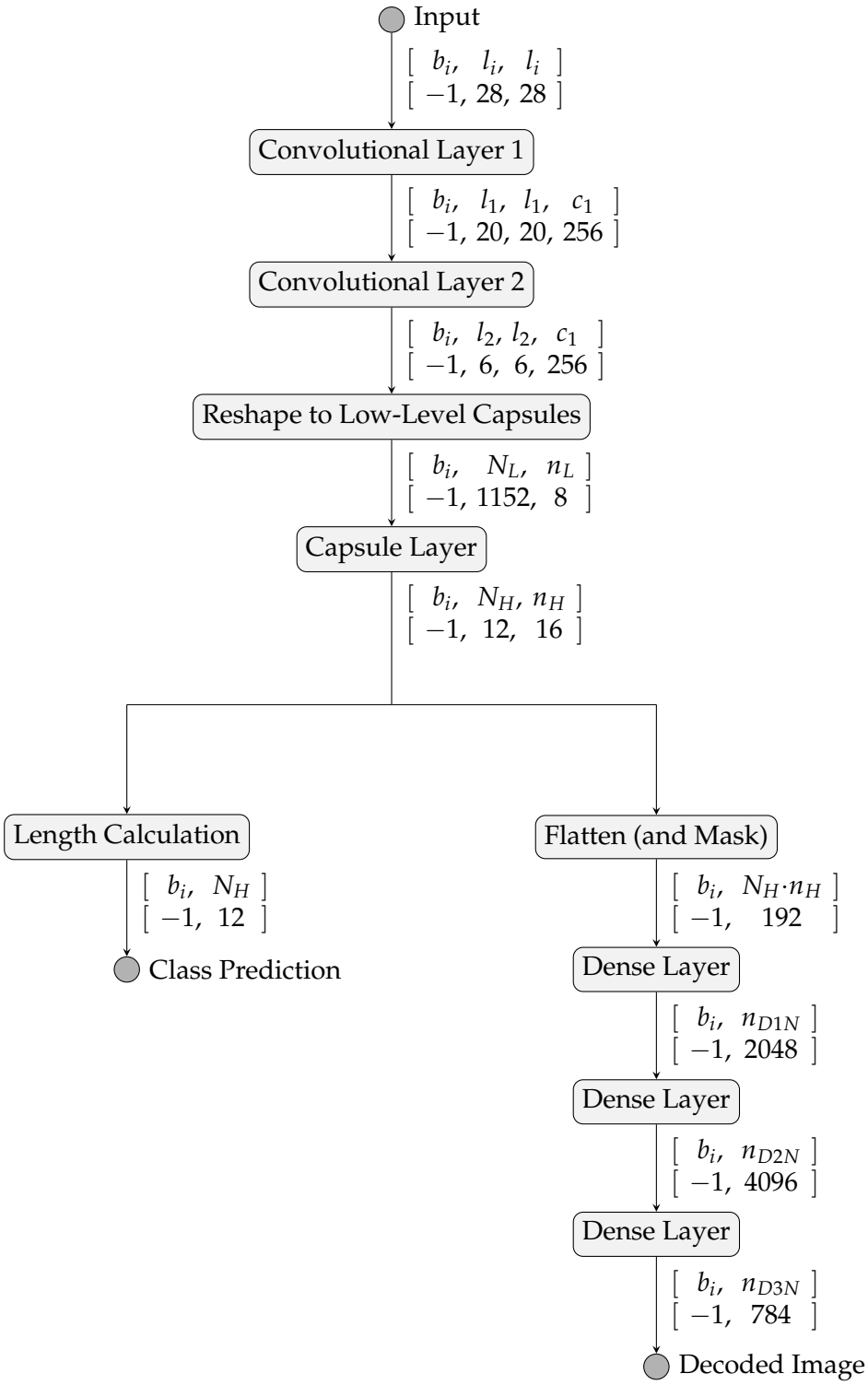
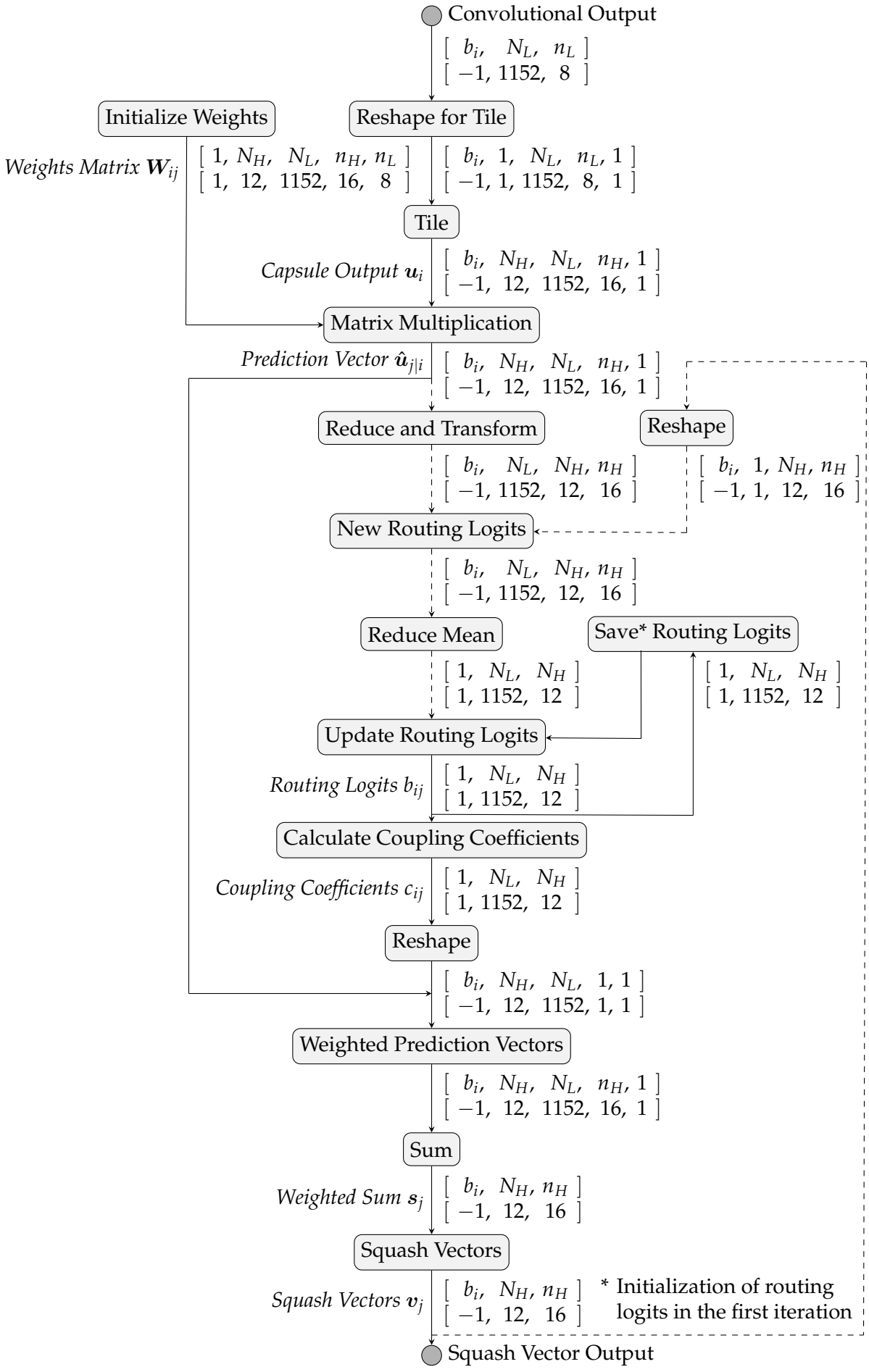


Figure 4.2: Calculation of tensors and their corresponding dimensions: General dimension for the upper square brackets, specific dimensions for the introduced case at the lower square brackets.

Figure 4.3: (subsequent page) Calculation of tensors and their corresponding dimensions. General and specific tensor information inside the brackets equivalent to figure 4.2. Solid connections indicate the execution before dashed connections.


Convolutional Output
[b_i, N_L, n_L]
[−1, 1152, 8]
Initialize Weights
Reshape for Tile
Weights Matrix $\boldsymbol{W}_{ij}$
[1, N_H, N_L, n_H, n_L]
[1, 12, 1152, 16, 8]
[b_i, 1, N_L, n_L, 1]
[−1, 1, 1152, 8, 1]
Tile
Capsule Output $\boldsymbol{u}_i$
[b_i, N_H, N_L, n_H, 1]
[−1, 12, 1152, 16, 1]
Matrix Multiplication
Prediction Vector $\hat{\boldsymbol{u}}_{j|i}$
[b_i, N_H, N_L, n_H, 1]
[−1, 12, 1152, 16, 1]
Reduce and Transform
Reshape
[b_i, N_L, N_H, n_H]
[−1, 1152, 12, 16]
[b_i, 1, N_H, n_H]
[−1, 1, 12, 16]
New Routing Logits
[b_i, N_L, N_H, n_H]
[−1, 1152, 12, 16]
Reduce Mean
Save* Routing Logits
[1, N_L, N_H]
[1, 1152, 12]
[1, N_L, N_H]
[1, 1152, 12]
Update Routing Logits
Routing Logits b_{ij}
[1, N_L, N_H]
[1, 1152, 12]
Calculate Coupling Coefficients
Coupling Coefficients c_{ij}
[1, N_L, N_H]
[1, 1152, 12]
Reshape
[b_i, N_H, N_L, 1, 1]
[−1, 12, 1152, 1, 1]
Weighted Prediction Vectors
[b_i, N_H, N_L, n_H, 1]
[−1, 12, 1152, 16, 1]
Sum
Weighted Sum $\boldsymbol{s}_j$
[b_i, N_H, n_H]
[−1, 12, 16]
Squash Vectors
Squash Vectors $\boldsymbol{v}_j$
[b_i, N_H, n_H]
[−1, 12, 16]
* Initialization of routing logits in the first iteration
Squash Vector Output

The input of the capsule-decoder-network is a three-dimensional input tensor that contains one batch of greyscale images. The outputs of the capsule-decoder-network are the lengths of the squash vectors in form of a 12-dimensional vector and the flattened decoded image. The input and output are illustrated in figure 4.2 as grey circles.
The implementation of the model is done by the model class of Keras. The layers of figure 4.2 are defined in the model's constructor and the transfer of the tensors from one layer to the next is defined in the model's .call() method. Herein, for each layer, the incoming and outgoing tensor is set, equivalent to Keras's functional API. [31]
The convolutional layers are defined by the parameters noted in figure 4.2. A bias is used in both layers. The first convolutional layer is activated by the ReLU function [26] while for the second layer the leaky-ReLU [32] function is used for activation. Without the leaky-ReLU activation function, large scopes of zeros were produced frequently, that led to a vector with a length of 0, that subsequently led through calculation errors by attempting the division by 0.
The model class provides the possibility for the definition and usage of custom functions as layers within the .call() method. These so-called 'Lambda' layers are recommended to wrap simple operations in a Keras layer. [33] Thus, they are used for the implementation of the layer 'Reshape to Low-Level Capsules' and the 'Length Calculation'. [31]
The layer 'Reshape to Low-Level Capsules' contains the modification of the incoming tensor's dimension with the .reshape() function provided by Tensorflow. [34] The output tensor of the second convolutional layer is converted into 1152 low-level capsules with each containing 8 neurons (figure 4.4).

$$[-1,\ 6,\ 6,\ 256] \leftrightarrow [-1,\ 6 \cdot 6 \cdot 32,\ 8] \leftrightarrow [-1,\ 1152,\ 8]$$

Figure 4.4: Illustration of the tf.reshape() function.

The layer 'Length Calculation' contains the application of the euclidean norm to the tensor to concatenate its 16-dimensional entries to one value per class.
The layer 'Flatten (and Mask)' uses all squash vectors a dimension of $[-1, 12, 16]$, which is subsequently called 'squash array'. If the capsule-decoder-network is training, all values except for those of the squash array are masked by setting their values to zero. To make this possible, the labels are passed on directly to the masking operations. The squash array is multiplied with the one-hot encoded labels, following that solely the values of the correct class remain. It is important to note, that the vectors are multiplied to keep

the gradients of the network. It is impossible, for example, to copy the correct squash array into another tensor containing only zeros. In this case, Keras will not be capable to calculate the gradients and, as a consequence, from the first layer of the network up to this layer, the weights will remain constant. Independent of training or testing, the masked or unmasked squash array is flattened eventually in this layer.

The layer 'Capsule Layer' contains the coupling between the low-level capsules and the high-level capsules. This layer fulfills two purposes: It creates the correct weights connections between the capsules and it implements the dynamic routing algorithm. This layer requires the initialization and training of weights. A Lambda layer does not contain trainable parameters, and thus a capsule layer cannot be implemented by the usage of a Lambda layer.

The alternative for a Lambda layer, that supports trainable parameters, is provided by Tensorflow through subclassing the tf.keras.layers.Layer class. [35] The capsule layer inherits from this class and is called inside the model of the capsule-decoder-network. The operations included in the capsule layer are visualized in figure 4.3 on page 34. Similar to the previously discussed figure, the operations in figure 4.3 are illustrated by grey-colored fields. These fields do not represent layers but rather operations between the tensors that are connected by the arrows. Next to each arrow, the general and specific dimension of the corresponding tensor is noted. The names of the specific tensors are written in italic text to the arrows. Those tensors and their dimensions in the implementation are additionally summarized in table 4.1.

Table 4.1: Overview of the general and specific dimensions of the named tensors inside the capsule layer visualized in figure 4.3

Parameter Name	**Parameter Symbol**	**General Dimension**	**Specific Dimension**
Low-Level Capsule Output	$\boldsymbol{u}_i$	$[b_i, N_H, N_L, n_L, 1]$	[-1, 1, 1152, 8, 1]
Weight Matrix	$\boldsymbol{W}_{ij}$	$[1, N_H, N_L, n_H, n_L]$	[-1, 16, 1152, 12, 8]
Prediction Vector	$\hat{\boldsymbol{u}}_{j\|i}$	$[b_i, N_H, N_L, n_H, 1]$	[-1, 1, 1152, 16, 1]
Routing Logits	b_{ij}	$[b_i, N_L, N_H]$	[-1, 1152, 12]
Coupling Coefficients	c_{ij}	$[b_i, N_H, N_L]$	[-1, 12, 1152]
Weighted Sum	$\boldsymbol{s}_j$	$[b_i, N_H, n_H]$	[-1, 12, 16]
Squash Array	$\boldsymbol{v}_j$	$[b_i, N_H, n_H]$	[-1, 12, 16]

The input of the capsule layer is received by the low-level capsules. In this application, there are in total 1152 low-level capsules with each of them containing 8 neurons. The implementation of a capsule layer is performed in two steps:
First, the weights between the capsules must be set, because they are necessary for the dynamic routing. Secondly, for the implementation of the dynamic routing not only the tensors of the current iterations are used, but also values from the previous iteration.

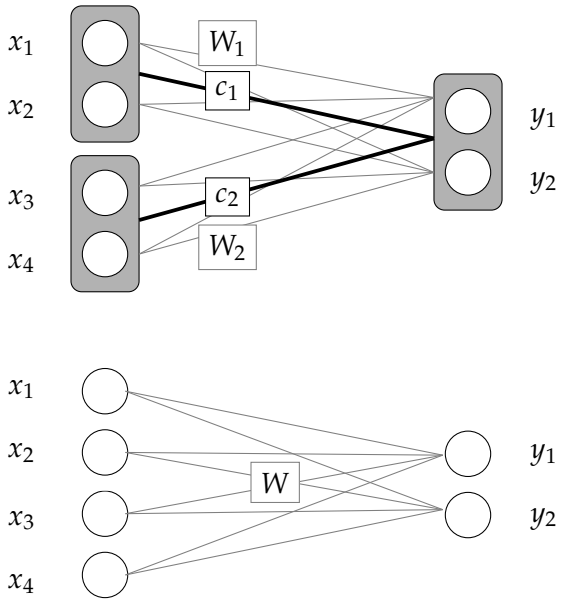


Figure 4.5: Example capsule layer and example dense layer

For the implementation of the weights between the low-level capsules and the high-level capsules the capsule layer is easily misinterpreted as a dense layer. Despite the similarity to a dense layer, there is an important difference to a capsule layer. The description of the difference is supported by figure 4.5 that related to equation 4.1 to equation 4.2. In a dense layer all values arriving in one neuron are summarized to one value (equation 4.1).

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 \cdot w_{11} + x_2 \cdot w_{12} + x_3 \cdot w_{13} + x_4 \cdot w_{14} \\ x_1 \cdot w_{21} + x_2 \cdot w_{22} + x_3 \cdot w_{23} + x_4 \cdot w_{24} \end{bmatrix} \tag{4.1}$$

In equation 4.1, y_1 and y_2 are the scalar values of each output neuron of a $[4, 2]$ dense layer. The parameter x_k are the scalar values of each input neuron k. There is exactly one connection between an input neuron and an output neuron m, which is described by the parameter $w_{k,m}$. There are two characteristics of the dense layer: Firstly, the scalar output values y_m are independent from each other. Secondly, for the calculation of the output scalar, all values are summed up after the multiplication with the weights $w_{m,k}$.
In a capsule layer the values are weighted by the coupling coefficients and subsequently summarized (equation 4.2). Furthermore, the prediction vectors are calculated for each connection between a low-level capsule and an high-level capsule. Therefore, the values

of different low-level capsules must not be concatenated (equation 4.3).

$$y = \begin{bmatrix} c_1 \cdot (x_1 \cdot w_{11} + x_2 \cdot w_{12}) & c_2 \cdot (x_3 \cdot w_{13} + x_4 \cdot w_{14}) \\ c_1 \cdot (x_1 \cdot w_{21} + x_2 \cdot w_{22}) & c_2 \cdot (x_3 \cdot w_{23} + x_4 \cdot w_{24}) \end{bmatrix} \tag{4.2}$$

$$y = \begin{bmatrix} \begin{bmatrix} y_{11} \\ y_{21} \end{bmatrix} & \begin{bmatrix} y_{11} \\ y_{22} \end{bmatrix} \end{bmatrix} \tag{4.3}$$

The modification from equation 4.1 to equation 4.2 is the application of the coupling coefficients c_i for each low-level capsule i. It shows that the single column of equation 4.1 is split into two columns and the term inside each column is weighted differently. In the application, the weight matrix between the neurons of the low-level capsules and the high-level capsules would have to be split into two separate vectors (equation 4.3). Therein, $y_{i,m}$ is the prediction for each low-level capsule i. This cannot be performed with the standard dense layer of Keras, because this layer uses the calculation of the output scalars like equation 4.1. Therefore, the specific arrangement of the capsules is implemented inside the capsule layer. The network is 'decreased' virtually to this point at which a dense layer is simulated. The smallest unit applicable for a dense layer is a connection between one low-level capsule with one high-level capsule. To connect each low-level capsule with each high-level capsule without mixing the results of the matrix multiplications, the low-level capsules are 'copied' one time for each high-level capsule. This is done in Tensorflow by the function tf.tile(), that creates a tensor by replicating the provided tensor for specific amounts along all of its axes. [36]. This method allows the reproduction of values along one or multiple axes which is not possible by comparable functions like tf.stack() [37] or tf.expand() [38]. An example of the application of the tf.tile() function is provided by figure 4.6.

$$\begin{bmatrix} [1\ \ 1] & [2\ \ 2] \end{bmatrix} \xrightarrow{\text{tf.tile [2, 3]}} \begin{bmatrix} [1\ 1\ 1\ 1\ 1\ 1] & [2\ 2\ 2\ 2\ 2\ 2] \\ [1\ 1\ 1\ 1\ 1\ 1] & [2\ 2\ 2\ 2\ 2\ 2] \end{bmatrix}$$

Figure 4.6: Illustration of the tf.tile() function. [36] The two-dimensional tensor of the left side is reproduced two times along the first (outer) axis and three times along the second (inner) axis.

In reference to figure 4.3, the dimensions of the incoming tensor of low-level capsules is increased ('Reshape for Tile'), so that one set of all low-level capsules is created for all

12 high-level capsules. Note, that in figure 4.3, the second dimension of the tensor is 1 before the 'Tile' operation and it is set to $N_{OC} = 12$ afterwards.

To connect the set of low-level capsules to the high-level capsules the weights are created. In subclasses of tf.keras.layers.Layer weights are initialized in the .build() method of the class. This method is called one time, when an instance of the class is created. [35]

The application of weights onto input values is achieved by a matrix multiplication. In Keras and Tensorflow, the last two-dimensions of both arrays are considered as input for the matrix multiplication, if not stated otherwise. [39] The dimensions of two tensors are compatible for a matrix multiplication if the second considered dimension of the first tensor is equal to the first considered dimension of the second tensor. All remaining dimensions must be either 1 or equal. The calculation for this example is shown in figure 4.3:

$$[1,\ 12,\ 1152,\ \underline{\underline{16}},\ \underline{8}] \times [-1,\ 1,\ 1152,\ \underline{\underline{8}},\ \underline{1}] \rightarrow [-1,\ 12,\ 1152,\ \underline{\underline{16}},\ \underline{1}]$$

Figure 4.7: Illustration of the tf.linalg.matmul() function. [39] The symbol '$\times$' illustrates the operation. The dimensions considered for the matrix multiplication are underlined twice

The result of the matrix multiplication and the low-level capsules is the prediction vector $\hat{\boldsymbol{u}}_{j|i}$. It has two application cases that are executed subsequently. In figure 4.3, the operations connected by the solid lines are performed before the operations connected by the dashed lines. The output produced by the solidly connected lines is the squash array. The operations connected by the dashed lines are the update operations of the routing algorithm and by them, new routing logits are calculated.

The first time this cycle is passed, no routing logits exist. They are initialized with zeros by the step 'Save* Routing Logits'. They are passed through the operation 'Update Routing Logits' without any modification, because there is no value to update them and then they are used to calculate the coupling coefficients. Deviating from the network structure in [16], the routing logits are normalized to values between 0 and 1 in this implementation. The reason is, that an unlimited rise of routing logits and coupling coefficients was observed to the point that *NAN* values were generated. This behaviour was corrected by normalizing the routing logit values.

The coupling coefficients are multiplied with the prediction vector ('Weighted Prediction Vectors') and added ('Sum'). In the operation 'Squash Array' the squash function (equa-

tion 3.4) is applied. The required euclidean norm of the weighted sum is calculated by Tensorflow's function tf.norm(). [40]

As soon as a squash array is present, the routing coefficients for the next iteration are calculated. The squash array is reshaped and multiplied with the prediction vector ('New Routing Logits'). To this point, the routing logits would be different for each output neuron and they would be different for each image inside the batch. To balance the routing logits, the mean is calculated along the dimension of all output neurons n_H and along the batch size b_i ('Reduce Mean'). The result is an array that contains $N_{PC} = 1152$ routing logits for each of the $N_{OC} = 12$ high-level capsules. The routing logits are stored in a .npy-file and in all subsequent steps they are loaded instead of initialized ('Save Routing Logits').

To examine the capsule-decoder-network for its applicability for image rankings, and to verify its functionality, the tensors of the capsule-decoder-network must be available in a human readable form. This is guaranteed by Tensorflow's 'Eager Mode'. [41] The Eager Mode is a comparably new mode that is used as the default mode since the update from Tensorflow 1.x to Tensorflow 2.x. The mode, that was formerly used in Tensorflow 1.x versions, is the 'Graph Mode'. [42] The main difference according to [41, 42] is the process in which operations of tensors are executed: In the Graph Mode a 'Tensorflow graph' is constructed and executed. All calculations are performed inside the graph, so that only the input and outputs are available for the user. Instead, in the Eager Mode, all operations are executed by Python. As a consequence, not only the input and output tensor, but all involved tensors are easily accessible. Therefore, the Eager Mode is the preferred mode when debugging a network or when the accessibility of the involved tensors is necessary. In the implementation of this model, all tensors are storable within .csv or .txt files. Additionally, not only the values of the squash array are stored, but they are also decoded into storable images with the tf.io.write_file() function. [43] This feature is used in chapter 5 for the visualization of squash vectors.

4.2 Training of the Capsule-Decoder-Network

The model was trained on 16384 of the reduced EMNIST dataset containing handwritten digits. It was tested with 1024 images. [17] The test images were predicted during the training process to plot the the loss and the accuracy. However, this test image set was

not used to adjust the hyperparamters of the model. The training parameters were set on basis of [16] and subsequently varied. The best found parameter setting is summarized in table 4.2. It was found, that more routing iterations per training step led to a faster adjustment of the coupling coefficients to the dataset, but at the same time, the model tends to overfit more. Deviating from [16], the routing iterations are set to 5 and the weight of the decoder is set to 0.105. The decay of the learning rate is applied to avoid overfitting in the later epochs. For the decay of the learning rate the Tensorflow scheduler for the exponential decay was used with an exponential decay and a staircase function. [44]

Table 4.2: Summary of applied parameters to train the capsule-decoder-network

Training Parameter	**Value**
Epochs	50
Batch Size	128
Routing Iterations	5
Learning Rate	10^{-4}
Decay Rate	0.96
Decay Steps	512
Decoder Weight	0.105

The capsule network and the decoder are concatenated into one model, because of the advantage of directly passing the squash array. As a result, three losses exist for the model: The first loss is the margin loss that is based on the margin loss function for the prediction of the label (equation 3.9). This loss is only applied to the capsule network and it is not impacted by the decoder. In figure 4.8a, the progress of this loss for the test dataset is plotted.

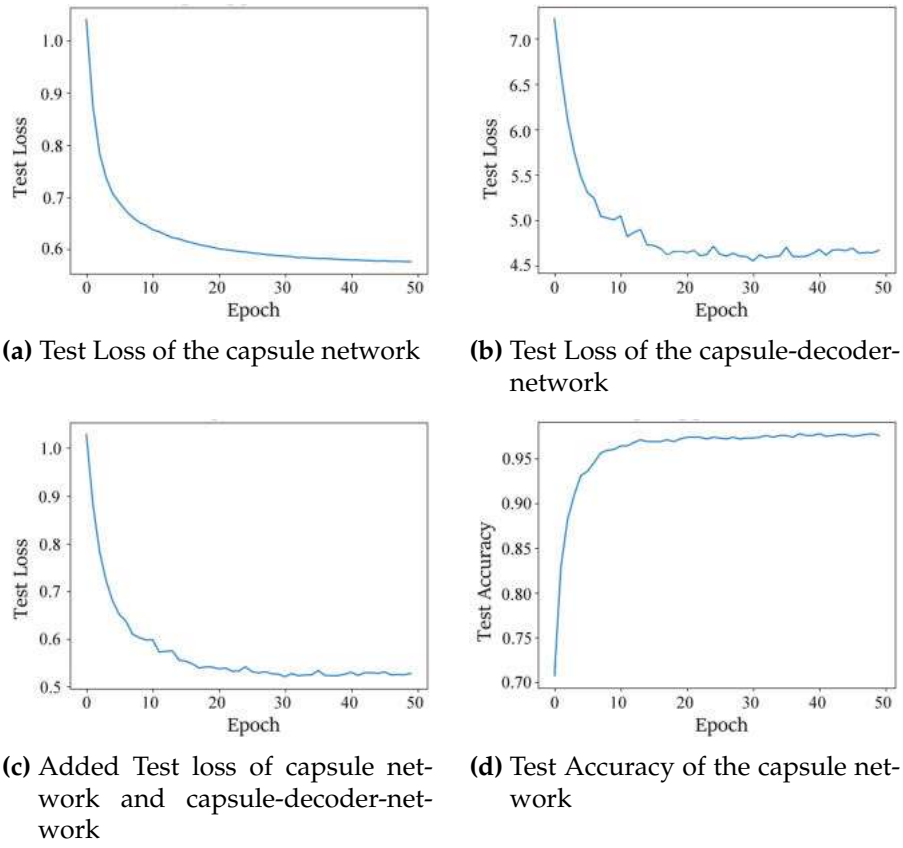
The second loss is the euclidean norm for the restoration of the image (equation 3.15). This loss is a result of the complete capsule-decoder-network. Its progress is plotted in figure 4.8b for the test dataset.

The total test loss L_{total} is calculated by the weighted sum of both test losses (equation 3.16).

$$L_{total} = L + d \cdot L_D \tag{3.16}$$

In this equation, L is the loss of the capsule network and L_D is the loss of the capsule-decoder-network. It is weighted by the decoder weight d because it is approximately one scale larger than the L_C. The losses are combined to encourage the high-level capsules to store features in their squash vectors. The combined test loss is shown in figure 4.8c.

The loss of the capsule network (figure 4.8a) decreases with a slightly higher gradient



(a) Test Loss of the capsule network

(b) Test Loss of the capsule-decoder-network

(c) Added Test loss of capsule network and capsule-decoder-network

(d) Test Accuracy of the capsule network

Figure 4.8: Plots of the training of the capsule-decoder-network.

than the loss for the capsule-decoder-network (figure 4.8b) because the optimization of the decoder relies on the optimization of the preceding parameters of the capsule network.

For the capsule network model the test accuracy reaches a value of 0.97. It is plotted in figure 4.8d. For the part of the decoder, no accuracy calculation is applicable because it predicts a pixel intensity and no label.

5 Creation of Image Rankings

The previous chapter covered the construction of the capsule-decoder-network and its training on 12 classes of the EMNIST dataset. This chapter covers the evaluation of this model for its applicability to the creation of image rankings.

For each of the 12 considered EMNIST classes a test dataset of 256 randomly selected images is created. Each of those class specific test sets is predicted by the capsule-decoder-network. During the prediction process the squash vector of each class from each test image is stored. The class, that is shown in the image, is subsequently referred to as 'present class' while all remaining classes are called 'non-present' classes.

Initially the relation between the squash vectors length and the quality of the image from the decoder is explored. This image is subsequently referred to as 'decoded image'. The squash array is inserted in two different modes into the decoder: In the first mode, the decoded images are generated based on the unmasked squash array. This mode is equal to the test mode, that was applied for the evaluation of the model's loss. In the second mode, a masked squash array is transferred to the decoder. In this squash array, only the squash vector of the present class carries nonzero values. This mode is equal to the masking applied in the training of the capsule-decoder-network. The relation between the length of the present's class squash vector and the restoration quality of the decoded image is examined. Therefore, the length of the squash vector of the present class is applied to rank a set of test images within each class (section 5.1).

The length of the squash vectors of non-present classes have an additional impact on the decoded image. To explore their impact, the squash array is masked differently, so that only one squash vector of a non-present class stays unmasked. All remaining squash vectors, including the one for the present class, are masked. By inserting this modified squash array into the decoder, the features are visualized, that the capsule-decoder-network assigned to the non-present class. The resulting images are provided in section 5.2.

For each class specific test image set, an overview of the largest non-present squash vectors is created. It summarizes the number of instances, that one specific non-present class generated the largest non-present squash vector of an image. The certainty for the non-present class is evaluated via the length of its squash vector. These overviews demonstrate the frequency of the largest non-present squash vector and thus, provide an insight to the perception of the capsule-decoder-network.

All overviews are summarized in a matrix similar to a confusion matrix. Therein, the number of instances of the largest non-present squash vector is noted for each class. To find out more about the vision ability of capsule networks, three frequent combinations of classes are determined from this matrix. For each combination a set of 8 images is created, that contains the step-wise morphing from one class to the other. By this image set, the point, on which the perception of the capsule-decoder-network changes from one class to the next one, is explored based on the squash vector's length (section 5.3). By the change of the squash vector's length in relation to the presence of features, the importance of features to the capsule-decoder-network's perception is investigated.

To provide an impression about the appearance of squash arrays, an example for class 'A' is displayed in figure 5.1. Both images are the visualization of the complete $[12 \times 16]$-dimensional squash array. The image on the left side visualizes the unmasked squash array and the image on the right side visualizes a masked squash array. The rows of

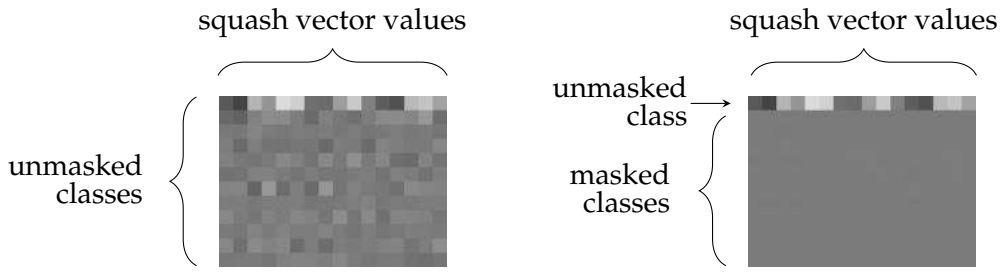


Figure 5.1: Visualization of a squash array for the recognition of class 'A'. Left: Unmasked squash tensor, right: Masked squash tensor except for class 'A'.
The color grey is assigned to the value 0. Lighter colors mean positive values, darker colors mean negative values.

both squash arrays are assigned to the 12 classes of the reduced EMNIST dataset, while the first row is assigned to class 'A' and the last row is assigned to class 'Z'. The columns show the values of the squash vector represented by a specific grey. To cover positive and negative values, the value 0 is represented by a medium grey and all positive values are displayed in lighter color and all negative values are displayed darker. The color referred to 0 is exactly the color of the masked area of the right squash array.

The certainty for a class is calculated by the euclidean norm of the corresponding squash vector. (equation 3.15) [28]

$$||\boldsymbol{v}_j|| = \Big(\sum_{p=1} \boldsymbol{v}_{j,p}^2 \Big)^{\frac{1}{2}} \tag{3.15}$$

In this equation, $v_{j,p}$ are the values of the squash vector $\boldsymbol{v}_j$. Because of squaring each $v_{j,p}$, both positive and negative values lead to the increase of the squash vector length. In the unmasked squash array of figure 5.1, the first row has the most light and dark pixels. This row assigned class 'A' is the prediction of the capsule-decoder-network based on the squash vector's length. Without knowing the exact values, the prediction is roughly assessable through the squash array. In this case, the actual result of the squash vector length for class 'A' is 0.90.

Additionally, row 2 for class 'B' and row 7 for class 'O' have several slightly lighter and darker pixels than the remaining rows for the non-present classes. The length of their squash vectors are 0.26 and 0.25, respectively. Because the loss function targets values for non-present classes below 0.10, a length higher than this value means a detection of elements of this class to a certain degree. If this squash array is inserted into the decoder, all squash vectors impact the resulting decoded image.

From a masked squash array as shown on the right side of figure 5.1, only the features of the unmasked class remain. Thereby, the part of the image is reconstructed by the decoder, that contributed to this specific class.

5.1 Image Rankings with Squash Vectors of the Present Class

The squash array is used for two different applications in the capsule-decoder-network: On one hand it is used for the prediction of the correct class. Therefore, the euclidean norm is calculated, which is used as measurement for the certainty for this class. On the other hand, the squash array is used as input for the decoder to reconstruct the image.

To evaluate the squash vector as measurement for the certainty for the prediction, for each test set of 256 images of one class, the distribution of squash vector length's between 0 and 1 is plotted. The plot for class 'A' is shown in figure 5.2. The plots for the remaining classes are shown from figure A.1 on page 71 to figure A.11 on page 74.

The length of the squash vectors are grouped by their values in 0.05-steps on the horizontal axis. The number of images in this interval is noted on the vertical axis. For all classes, the majority of squash vectors has a length between $0.70 \leq ||v_A|| \leq 0.95$. This distribution fits to the achieved accuracy of 0.97 with the test dataset (section 4.2). Because the class of the largest squash vector leads to the prediction, based on the length distribution the majority of squash vectors is predicted correctly. It could be argued, that

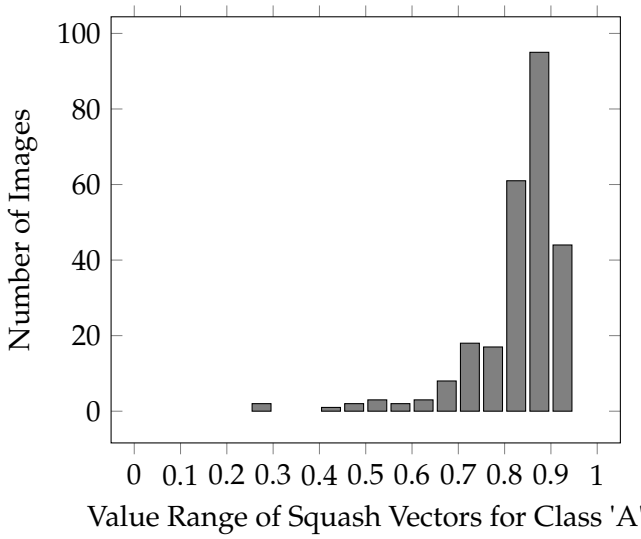


Figure 5.2: Distribution of squash vectors from 256 test images containing class 'A'

the length of the non-present squash vectors are unknown in this distribution, so that possibly another larger squash vector for another class could exist. However, it will be shown in section 5.2 that in no case two large squash vectors ≥ 0.7 exist.

The length of the squash vectors seems reasonable under the aspect of the application of the margin loss function (equation 3.9 on page 20). Therein, a loss of 0 is generated if the squash vector of the present class has a length ≥ 0.9. However, the growth of the squash vector length is not encouraged above the value of 0.9. This explains why the squash vector's increase stops shortly above 0.90.

Before using the squash vector as base for the ranking, the relation between its length and the quality of the decoded image is examined. The image's quality is measured by the structural similarity (SSIM) between the decoded image I_1 and the original image I_2 (equation 5.1). [45]

$$\mathrm{SSIM}(I_1, I_2) = \frac{(2\mu_1\mu_2 + c_1)(2\sigma_{12} + c_2)}{(\mu_1^2 + \mu_2^2 + c_1)(\sigma_1^2 + \sigma_2^2 + c_2)} \tag{5.1}$$

The parameter I_i is the average of each considered image I_i, the parameter σ_i^2 is the variance of each image I_i and the parameter σ_{12} is the covariance of I_1 and I_2. The additional constants c_i are applied to stabilise the division by a weak denominator.

This calculation is performed for the decoded images based on the masked and the unmasked squash array, respectively. In this section, the term 'masked' always refers to the masking of the squash vectors of the non-present classes, so that solely the values of the

present squash vector remain in the squash array.

A correlation between the squash vector length and the SSIM is found for images decoded from masked squash arrays: A larger squash vector leads to higher SSIM between the original image and the prediction. The plot for class 'A' is displayed in figure 5.3. For the remaining classes, the plots are shown in figure B.1 on page 75 and figure B.2 on page 76. For images decoded from the unmasked squash array no relation between the length of the squash vector and the SSIM was found.

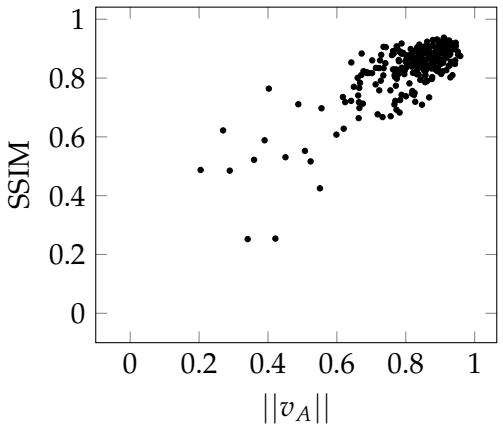


Figure 5.3: Plot for the lengths of the squash vectors for class 'A' and the resulting SSIM between the original image and the decoded image

Because there is a correlation for images of masked squash arrays, but no correlation for images of unmasked squash arrays, the length of the squash vector provides a measurement about the affiliation of a prediction to a specific class. This affiliation measurement is required for the creation of image rankings.

For all classes, a ranking is created for images restored from squash vectors in an interval between $0.70 \leq ||v_j|| \leq 0.99$. The ranking for images of class 'A' is illustrated in figure 5.4 on the following page. The length of the squash vector descends from a length of 0.99 to a length of 0.70 in steps of 0.01.

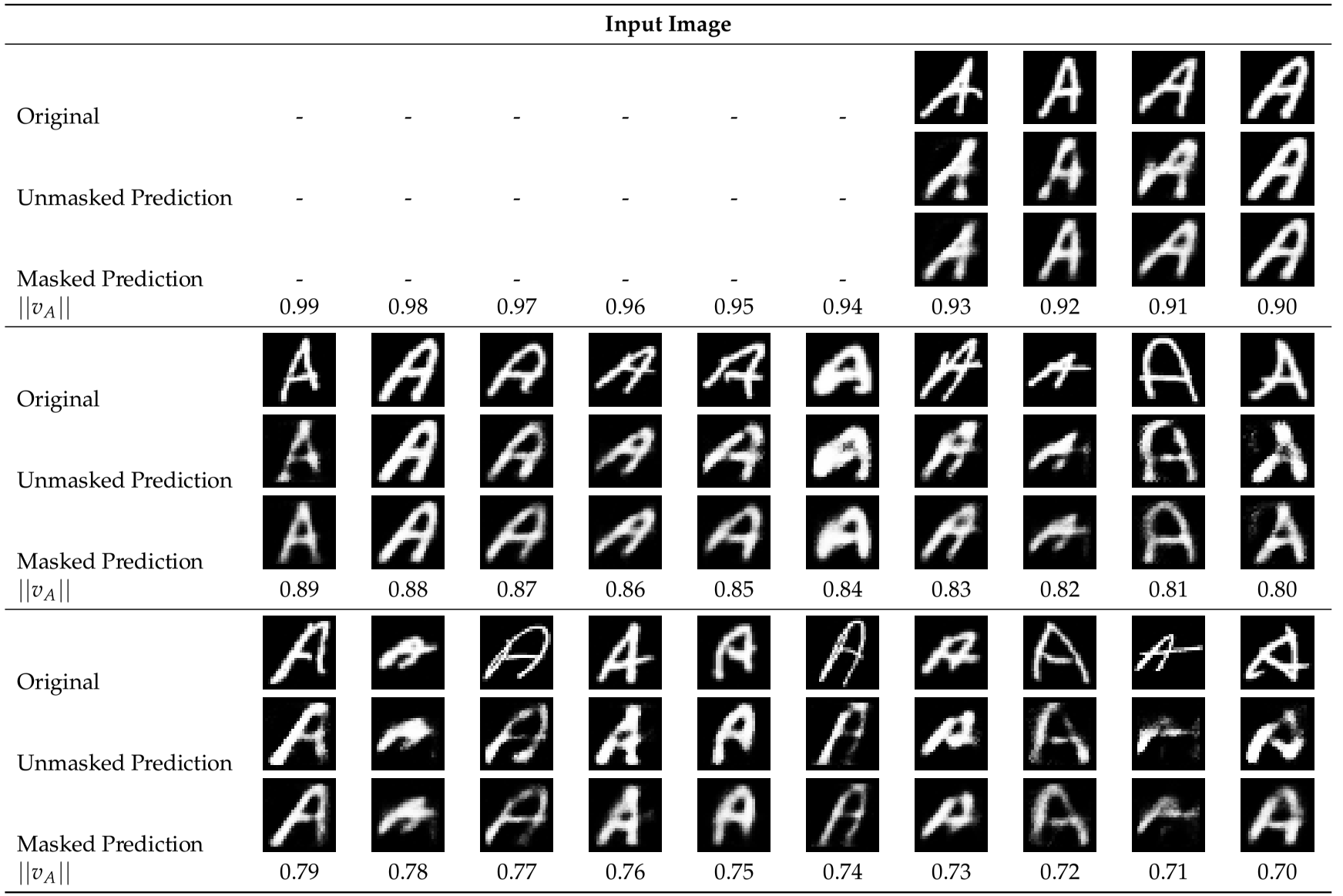


Figure 5.4: Ranking of images for class 'A' based on the length of the corresponding squash array for three intervals between 0.7 and 0.99.

For each step, the original image is provided at the top of a row, the decoded image from the unmasked squash array, is shown in the middle and the decoded image based on the masked squash array is shown in the bottom row. The prediction, that is produced from the masked squash array, is referenced as 'masked' prediction and the prediction based on the unmasked squash array is accordingly referenced as 'unmasked prediction'.
Equivalent overviews for the remaining classes 'B' to 'Z' are displayed from figure C.1 on page 78 (class 'B') to figure C.11 on page 88 (class 'Z'). In all rankings from class 'A' to class 'Z' for a squash vector length of ≥ 0.85 the masked and the unmasked decoded images strongly resemble the original image and each other. This fits to the result of the calculation of the SSIM above in figure 5.3. However, the masked predictions appear smoothed in comparison to the originals. If the original images contain additional details, for instance a particularly short or large line, these details are restored only to a very low degree in the masked predictions.
The smoothing of the decoded figure is not found in the unmasked predictions. Instead, the images tend to have more irregularities as bolder or thinner lines and darker or lighter areas. This phenomenon increases with the descend of the squash vector length. A lower squash vector seems to be generated if the figure of the original image deviates significantly from the figures that generated a squash vector of ≥ 0.90. However, some details are restored to a larger degree in the decodings of the unmasked squash arrays in comparison to their masked counterparts.
These missing details in the images decoded from the masked squash array indicate that the capsule network tends to learn generalized representation of the class. One reason for that probably lays in the comparably large $[9 \times 9]$ convolutional filters that are applied in two convolutional layers previous to the capsule network. These filters strongly concatenate features. Small features are unlikely to persist through two convolutional layers.
The images decoded from the masked squash array contain the features, that contributed to the prediction of the class. An image that generates a large squash vector has more features that contribute to the prediction of a class than a shorter squash vector. However, it is remarkable, that not all features from the decodings of the masked squash array are also found in the decoded image of the unmasked squash array. This impact of the squash vectors of non-present classes is explored and described in the subsequent section 5.2.

5.2 Impact of Non-Present Squash Vectors on the Decoded Image

As shown in the previous section, the length of the squash vector for the present class correlates with the quality (SSIM) of the decoded image for the present class. Because there is no correlation between the length of the squash vector and the predictions based on the unmasked squash array, an impact of the non-present classes on the output image is assumed.

The evaluation of the squash vectors of the non-present classes can help to understand the perception of the capsule-decoder-network. Equivalent to the previous section, from figure 5.5 on page 53 to figure 5.7 on page 54 twelve images for the class 'A' are ordered based on the length of their correct squash vector. In this example, the squash vectors range from 0.25 to 0.90. Additionally, for each image the largest squash vectors among the non-present classes is found and their features were restored by the decoder. Equivalent examples are also produced for class 'C' and class 'K'. They are presented from figure D.1 on page 100 to figure D.6 on page 103.

In first row of each table, the original image is displayed. In the second row the prediction based of the unmasked squash array is shown. As stated above, the unmasked squash array contains the predictions for all classes.

In the third row the prediction for the present class 'A' is illustrated. These images are produced solely on the squash vector for class 'A' while the lengths of all squash vectors for the non-present class were set to zero. The predictions of figure 5.5 are properly restored letters. The quality and the intensity of the restoration decline as the squash vector for the correct class 'A' decreases. Especially the restored letters (A9 to A12) of figure 5.7 are not recognizable as complete letter. Only specific parts of the letter 'A' are restored.

The fourth row of figure 5.5 and the fourth and fifth row of figure 5.6 and figure 5.7 contain decoded images that were made based of other squash vectors of non-present class in the specific image. For each image, the length and the class of its largest squash vector among the non-present classes is provided. For example, for image A1 the squash vector for class 'O' has a length of 0.25. Despite the large squash vector for the correct class 'A' is $||v_A|| = 0.90$, the decoded image of the unmasked squash array is impacted by this short squash vector too. In this case, the image for 'O' has an arc on the top which is not found in the decoded image for 'A' based on the masked squash tensor but it is found in the decoded image from the unmasked squash tensor. This indicates a superposition of

the decodings of all squash vectors.

For image A4 and A5 in figure 5.6, the decodings from the next two largest squash vectors of non-present classes are illustrated. The reason for this is, that the lengths of their second largest incorrect squash vectors are still comparably high to the remaining squash vectors. The capsule-decoder-network sees in image A4 not only features of the class 'A' but also features of class 'Q' and features of class 'N'. The superposition of this letters still produce the image of an 'A' at the output but especially the 'A's right line is not reconstructed properly despite seen the masked 'A'. Because it was not recognized for the 'Q' and for the 'N' the intensity decreases to a point, by which it becomes smaller than it would have been in comparison to the pure decoding of the masked squash array for class 'A'.

The same phenomenon occurs to a greater extend for image A5. In this case, the 'A' is recognized properly, but in the image of the unmasked squash array is overlaid by the recognition of the class 'K' and the class 'X'. In comparison to the preceding letter A4, in this case the squash vectors for the correct and incorrect class are closer to each other: $||v_A|| = 0.69$ compared to $||v_K|| = 0.51$. This leads to an higher impact of class 'K' onto the decoded image. As a consequence, the letter 'A' is not decoded completely and the upper right slash of the 'A' is missing. A smaller difference between the length of the squash vectors seems to lead to a stronger superposition in the decoded image. This superposition could be the reason for the restoring of some details which are not predicted by the correct class: If a detail of a letter is assigned to a different class with a sufficient large squash vector, this detail is also restored in the output image.

The behaviour of the letters in figure D.3 is equivalent. Therein, the letters are almost not recognizable but their particular features are comparably well restored by the superposition of multiple decoded images of low squash vectors. Likewise, for class 'C' and class 'K', a superposition of the decoded images of large squash vectors is observed. One remarkable example of the restoration of an incorrect feature is shown in figure D.3 for image A9. Although the 'C' of the original image is not properly reconstructed by the prediction of its class, it is partially restored by the prediction of class 'R'. A similar effect takes place in figure D.6 for letter A2. The serif-like line on the right top of the 'K' is interpreted as feature of the class 'C' but not of class 'K'. Because the length of the squash vector for class 'C' is comparably high, this detail is found in the decoded image of the

unmasked squash array.

Another remarkable effect is found in figure 5.7: While the first masked prediction of A9 was made based on a squash vector length of $||v_C|| = 0.51$ its decoded image is significantly worse than the comparable one for A12, which has a squash vector length of $||v_C|| = 0.38$. The reason for this lays probably in the single values of both squash vectors: The decoder learns to restore images from a specific search space and the weights are adjusted on it. If the single values do not fit into the search space, the decoding quality is low in comparison to the large squash vector. In the opposite case, if a short squash vector has features that are part of the search space, the restored image gets a higher quality. This also explains the comparatively large variation of SSIM results for squash vectors of similar length (figure 5.3).

In conclusion, the squash vector supports the separation of the capsule-decoder-network's perception for each class. The lager the squash vector of a class, the higher is its impact on the decoded image.

Independent from the affiliation to the present or non-present class, all squash vectors have in common that the rise of the length leads to an increase of the intensity and clarity of the image. The threshold for clear letters seems to be around a squash vector length of 0.70. Despite that, there is also a difference in the clarity between decodings from a squash vector length of 0.70 and 0.90.

The squash vectors of the non-present classes have a significant impact on the decoded image. The areas of the decodings of non-present squash vectors give an insight which features are perceived by the capsule-decoder-network: These features help to explain which areas are misunderstood from the capsule-decoder-network and how they were interpreted. It is possible to tell the certainty and the area for the correct class and simultaneously the classes that are also perceived by the capsule-decoder-network.

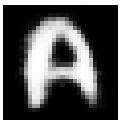
Image Number	A1	A2	A3	A4
Original				
Unmasked Prediction				
Masked Prediction				
$\|\|v_{present}\|\|$	$\|\|v_A\|\| = 0.90$	$\|\|v_A\|\| = 0.83$	$\|\|v_A\|\| = 0.82$	$\|\|v_A\|\| = 0.81$
Masked Prediction				
$\|\|v_{nonpresent}\|\|$	$\|\|v_O\|\| = 0.25$	$\|\|v_B\|\| = 0.42$	$\|\|v_O\|\| = 0.43$	$\|\|v_B\|\| = 0.31$

Figure 5.5: Comparison of masked predictions and unmasked predictions for class 'A' and the predictions from 'non-present' squash vectors

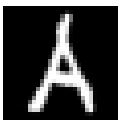
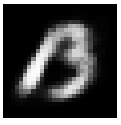
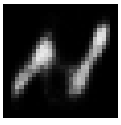
Image Number	A4	A5	A6	A8
Original				
Unmasked Prediction				
Masked Prediction				
$\|\|v_{present}\|\|$	$\|\|v_A\|\| = 0.71$	$\|\|v_A\|\| = 0.69$	$\|\|v_A\|\| = 0.68$	$\|\|v_A\|\| = 0.68$
Masked Prediction				
$\|\|v_{nonpresent}\|\|$	$\|\|v_Q\|\| = 0.42$	$\|\|v_K\|\| = 0.54$	$\|\|v_B\|\| = 0.54$	$\|\|v_N\|\| = 0.53$
Masked Prediction				
$\|\|v_{nonpresent}\|\|$	$\|\|v_N\|\| = 0.27$	$\|\|v_X\|\| = 0.33$		

Figure 5.6: Comparison of masked predictions and unmasked predictions for class 'A' and the predictions for non-present squash vectors

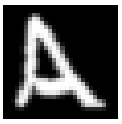
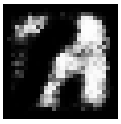
Image Number	A9	A10	A11	A12
Original				
Unmasked Prediction				
Masked Prediction				
$\|\|v_{present}\|\|$	$\|\|v_A\|\| = 0.62$	$\|\|v_A\|\| = 0.51$	$\|\|v_A\|\| = 0.49$	$\|\|v_A\|\| = 0.25$
Masked Prediction				
$\|\|v_{nonpresent}\|\|$	$\|\|v_X\|\| = 0.44$	$\|\|v_R\|\| = 0.26$	$\|\|v_S\|\| = 0.37$	$\|\|v_N\|\| = 0.35$
Masked Prediction				
$\|\|v_{nonpresent}\|\|$		$\|\|v_B\|\| = 0.24$	$\|\|v_C\|\| = 0.36$	

Figure 5.7: Comparison of masked predictions and unmasked predictions for class 'A' and the predictions from non-present squash vectors

For a large-scale estimation about the applicability of the usage of squash vectors of non-present classes, the following procedure is performed: For each class specific test set of 256 images, the largest squash vector of all non-present classes is determined. The class that corresponds to this squash vector and its length are stored. For each class the number of instances is counted in which each non-present class provided the highest squash vector. Additionally, the interval of all highest squash vectors of each non-present class is stored. Based on this data, an overview is created. The results for class 'A' are shown in table 5.1 on page 56. A comparable overview is created for the remaining classes 'B' to 'Z' from table D.1 on page 89 to table D.11 on page 99.

Therein, the results for the non-present squash vectors are additionally grouped in intervals of 0.1, which are provided in the left column of the table. The results for the non-present classes are shown in the remaining columns. Each entry for a non-present class consists of three values: The first value is the number of instances in the test dataset of 256 images, that this specific class provided the largest squash vector of a non-present

class. For instance, in the first row, the class 'B' generated the highest squash vector of a non-present class 15 times. Below the number of instances, the maximum and minimum of their squash vectors is noted. For the example of the non-present class 'B' in the first row, of all 15 these squash vectors, the largest has a length of 0.26 and the shortest had a length of 0.08.

In the majority of cases, the highest squash vector for the non-present class is smaller than the squash vector for the present class. Only for a few images, a low squash vector is generated for the present class and the squash vector of a non-present class reaches a higher value. In table 5.1, this occurs for squash vectors of class 'A' below a length of 0.2, as visible in the corresponding interval of $0.2 \leq ||v_A|| < 0.3$. At the bottom row of the table, the number of instances that this class generated the highest non-present squash vector is noted. For instance, class 'B' was found in 54 images among all 256 images generating the largest non-present squash vector.

Based on the ranking of figure 5.4, the more features the capsule-decoder-network finds for the present class, the larger the squash vector of the class. The same is true for the lengths of the non-present squash vectors. Therefore, the length of the squash vector of non-present classes provides insights to the model's perception. The minimum and maximum value in table 5.1 (and all corresponding tables for class 'B' to 'Z') provide information about the intensity of the detection of the non-present class. Based on the loss function, the length for a non-present squash vector is targeted to reach a value below 0.10. Therefore, if the length of the highest squash vector for a non-present class is < 0.10, the length of all other remaining squash vectors of non-present classes are even smaller. If simultaneously the length of the present squash vector is ≥ 0.90 the network solely perceives the present letter and nothing else. Even if the squash vector of the non-present class is slightly larger than 0.10 and the squash vector of the present class is slightly smaller than 0.90 the prediction of the capsule-decoder-network is still comparably unambiguous.

Table 5.1: Overview of the squash vectors lengths for the remaining classes in comparison to the length of the squash vector for class 'A'.

Squash Vector 'A' Length Interval	**Largest Squash Vector of Non-Present Class**										
	B	C	E	K	N	O	Q	R	S	X	Z
	Max. length of largest squash vector of non-present class Min. length of largest squash vector of non-present class										
$0.9 \leq \|\|v_A\|\| < 1.0$	15	2	5	4	3	1	0	11	1	2	0
	0.26	0.19	0.14	0.19	0.16	0.10	-	0.31	0.11	0.12	-
	0.08	0.15	0.08	0.12	0.12	0.11	-	0.10	0.11	0.10	-
$0.8 \leq \|\|v_A\|\| < 0.9$	29	6	4	11	17	7	12	36	4	21	9
	0.42	0.15	0.18	0.27	0.29	0.43	0.31	0.39	0.25	0.33	0.24
	0.09	0.10	0.08	0.12	0.09	0.11	0.10	0.11	0.10	0.11	0.12
$0.7 \leq \|\|v_A\|\| < 0.8$	7	0	0	2	7	2	6	9	0	2	0
	0.34	-	-	0.43	0.71	0.28	0.42	0.32	-	0.36	-
	0.14	-	-	0.32	0.17	0.21	0.13	0.19	-	0.30	-
$0.6 \leq \|\|v_A\|\| < 0.7$	2	0	0	3	3	0	2	0	0	1	0
	0.55	-	-	0.54	0.53	-	0.40	-	-	0.44	-
	0.37	-	-	0.22	0.24	-	0.35	-	-	0.44	-
$0.5 \leq \|\|v_A\|\| < 0.6$	0	0	0	1	0	0	1	2	0	0	1
	-	-	-	0.34	-	-	0.40	0.34	-	-	0.34
	-	-	-	0.34	-	-	0.40	0.32	-	-	0.34
$0.4 \leq \|\|v_A\|\| < 0.5$	1	0	0	0	1	0	1	0	0	0	0
	0.62	-	-	-	0.35	-	0.43	-	-	-	-
	0.62	-	-	-	0.35	-	0.43	-	-	-	-
$0.3 \leq \|\|v_A\|\| < 0.4$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
$0.2 \leq \|\|v_A\|\| < 0.3$	0	0	0	1	1	0	0	0	0	0	0
	-	-	-	0.41	0.37	-	-	-	-	-	-
	-	-	-	0.41	0.37	-	-	-	-	-	-
Sum of Instances	54	8	9	22	32	10	22	58	5	26	10

If the squash vector of a non-present class is significantly larger than 0.10, then features of this non-present class were detected in the image. Which elements of the present class impacted the non-present class is not apparent by the size of the squash vector. But in relation to figure 5.4 a larger squash vector is generated if more features of the class were found.

The difference between the length of squash vector of the present class and the length of the squash vector of the non-present class are decreasing for the length intervals of class 'A' in table 5.1. Although the squash vector of different classes are not coupled, it is

unlikely that features unique for one class are simultaneously unique for a second class. If the clarity of the features decreases, they are likely to be assigned to other classes.

Before the capsule layer is passed, the input image is processed by two convolutional layers, that 'prepare' the features for the low-level capsules. These features are ideally only concatenated into those low-level capsule that connect with the high-level capsule assigned to the present class. If features are concatenated into other low-level capsules than the length squash vector of the non-present class rises. If there is only one feature in a non-present low-level capsule, this is the single low-level capsule to reinforce the squash vector for the non-present class, so that its squash vector only rises to a small degree. This explains why large squash vectors do not tend to have other large squash vectors besides them.

The minimal length for squash vectors increases when the length of the squash vector for the present class is decreasing. This supports the thesis that more features are detected for non-present classes if less features are detected for the present class. This reaches a point by which the squash vector for the non-present class becomes larger than the squash vector of the present classes. In table 5.1 this point is reached for a squash vector of 'A' shorter than 0.3. Because the prediction of the capsule-decoder-network is based on the largest squash vector, for those images an incorrect class was predicted.

Generally, short squash vectors describe a low certainty of the model for the prediction. If another squash vector is higher than the capsule-decoder-network detected features for a different class, this class is detected instead of the correct class. If all squash vectors are short, the capsule-decoder-network does not find features assigned to any class.

In several cases, a substantial difference lays between the interval of the largest and smallest non-present squash vector for one class. For example, in table 5.1 within the interval $0.7 \leq ||v_A|| < 0.8$, the squash vectors for class 'N' range from 0.17 to 0.71. This means that in all those cases stronger evidence for class 'N' is found than for all other non-present classes. For the case of a squash vector of 0.71 these elements were comparably clearer than in the case for a squash vector of 0.17. In the bottom row of table 5.1 the relation between features of the present and the non-present class is apparent: In images of class 'A' features for class 'B' or class 'R' are recognized frequently. Very seldom, the features of the class 'C', 'E' and 'S' are found.

In table 5.1 (and in all other corresponding overviews for class 'B' to 'Z'), for each class

the number of instances that is generated the largest non-present squash vector is calculated. All the results are summarized in table 5.2 to provide an insight to the perception of the capsule-decoder-network. From this table it becomes apparent which features the capsule-decoder-network recognizes more or less frequently for a present class. Based on this matrix, the vision of the capsule-decoder-network is explored in more detail in the next section 5.3.

Table 5.2: Number of instances of the largest squash vector of non-present classes in comparison to the present class

Correct Class	**Number of Highest Squash Vector of Non-Present Class**											
	A	B	C	E	K	N	O	Q	R	S	X	Z
A	-	**54**	8	9	22	32	10	22	**58**	5	26	10
B	**45**	-	26	**44**	11	1	30	11	**37**	21	12	18
C	6	14	-	**43**	26	4	**62**	25	**30**	15	4	27
E	8	30	28	-	**43**	9	11	15	**44**	37	8	23
K	12	6	21	19	-	18	0	7	**90**	3	**72**	8
N	**51**	14	15	8	36	-	20	**42**	35	2	21	11
O	28	**40**	**53**	14	8	19	-	**46**	24	13	4	7
Q	27	22	22	6	10	30	**50**	-	**31**	**32**	9	17
R	22	26	27	9	**86**	19	2	27	-	4	17	17
S	6	**61**	10	39	11	12	17	34	25	-	25	16
X	27	6	17	15	**89**	18	2	13	20	21	-	28
Z	14	21	**43**	23	18	10	5	11	**53**	16	**42**	-

5.3 Image Rankings for Transformed Images of Similar Classes

In table 5.2 the classes which are misrecognized most frequently for a present class are displayed. Based on the table, the classes most frequently found within other classes are the combinations of class 'K' and 'R' (86 and 90 times), 'C' and 'O' (53 and 62 times). From the remaining class combination, the next frequent combination is 'A' and 'N' (32 and 51 times).

For these classes, the point at which the class label changes from one class to another is of great interest to comprehend the capsule-decoder-network's vision. For each combination, a set of 8 images is created, whose images are gradually morphed from one class of the combination to the other. The set for the classes 'K' and 'R' are shown in figure 5.8. The start and endpoint of the image set are images taken from the test dataset and have a squash vector length larger than 0.89. The artificially morphed images are created manually. The results for the predictions of the image sets by the capsule-decoder-network are

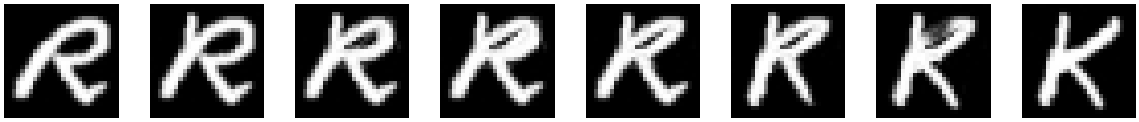


Figure 5.8: Morphed image set for the class 'K' and 'R'. The image on the left and on the right are unmodified images of the EMNIST test dataset. [17] The remaining images were created manually.

shown from figure 5.9 to figure 5.11 (page 60 to 62).

In figure 5.9 to figure 5.11, the unmasked squash arrays are visualized as well as three different predictions by the decoder: The prediction based on the unmasked squash arrays of column 3 is shown in column 4. In column 5 and 6, the predictions were produced for a squash vector that is masked except for the considered class. Below each image of column 5 and 6, the length of the squash vector of the corresponding class is noted.

The more dark and light pixels are contained in a squash vector, the higher is its length. Based on image 1 and 8, the position of the squash vector of the considered class is recognizable. With the modification of the images from one class to the other, the squash vector of the starting class fades gradually while the other one is rising.

The predictions, that were created with the masked squash array, appear 'smoothed' in comparison to the input image and to the unmasked prediction. Details that strongly deviate from the classic form of the class, for instance the small slash of class 'R' at images 1 to 5 in figure 5.9 or the slash of class 'A' in images 4 and 5 in figure 5.11 are neither restored in the masked nor in the unmasked prediction.

In comparison, at least one image created from the masked squash array appears of higher quality than the decoding from the unmasked squash array. The reason for that could be the previously described superposition of all vectors in the unmasked output image. The combinations of the single squash vector outputs seem to work based on the intensity, which seems to be related to the length of the squash vector. As a result, at the superposition of the restored areas, irregularities may appear in the decoded image as for example in image 4 of figure 5.9, in which the missing connection point of the letter 'K' is transferred to a missing connection point in the unmasked prediction image. This happens despite the large squash vector for the class 'R'.

In the transfer between two classes, each image set has one point, on which the length of the squash vector of is increased by ≥ 0.3 between one image and the next. This threshold could be explained by the application of the leaky-ReLU [32] function inside the convo-

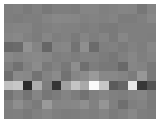
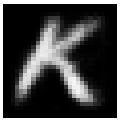
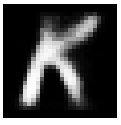
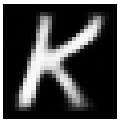
No.	Input Image	Squash Images	Reconstructed Images		
		Unmasked	Unmasked	Masked Except Class: 'K' $\|\|v_K\|\|$	'R' $\|\|v_R\|\|$
1				 0.07	 0.91
2				 0.12	 0.89
3				 0.18	 0.88
4				 0.18	 0.85
5				 0.41	 0.74
6				 0.5351	 0.7399
7				 0.78	 0.54
8				 0.90	 0.18

Figure 5.9: Predictions of the decoder for the image set containing morphed images between the classes 'K' and 'R'. Column 4 contains the reconstructed images for the unmasked squash vector of column 3. The images in column 5 and 6 were created by masking all squash vectors except for class 'K' or class 'R', respectively. The numbers below the images $||v_K||$ and $||v_R||$ are the lengths of the squash vector of the specific class.

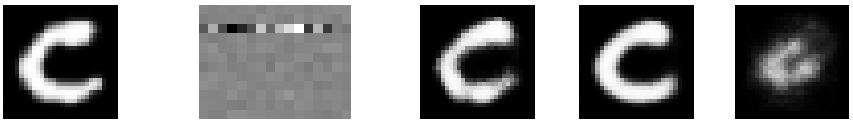
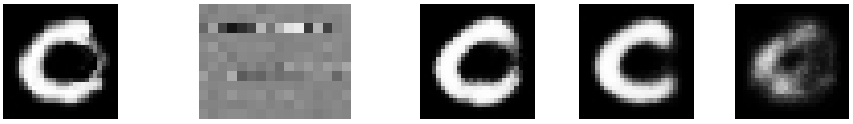
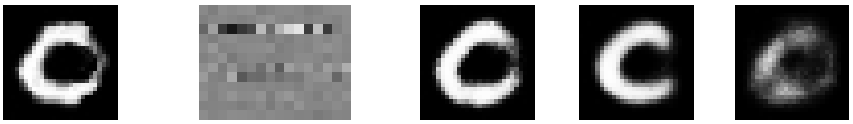
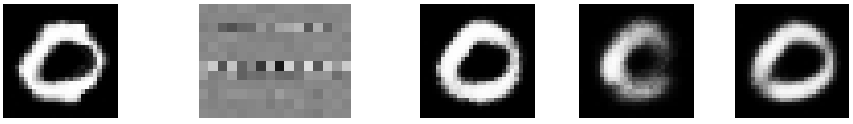
No.	Input Image	Squash Images	Reconstructed Images		
		Unmasked	Unmasked	Masked Except Class: 'C' $\|\|v_K\|\|$	'O' $\|\|v_R\|\|$
1				0.91	0.09
2				0.85	0.11
3				0.82	0.20
4				0.81	0.26
5				0.40	0.78
6				0.14	0.88
7				0.04	0.92
8				0.07	0.93

Figure 5.10: Predictions of the decoder for the image set containing morphed images between the classes 'C' and 'O'. Column 4 contains the reconstructed images for the unmasked squash vector of column 3. The images in column 5 and 6 were created by masking all squash vectors except for class 'C' or class 'O', respectively. The numbers below the images $||v_C||$ and $||v_O||$ are the lengths of the squash vector of the specific class.

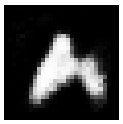
No.	Input Image	Squash Images Unmasked	Reconstructed Images Unmasked	Masked Except Class: 'A' $\|\|v_A\|\|$	Masked Except Class: 'N' $\|\|v_N\|\|$
1				 0.03	 0.92
2				 0.05	 0.91
3				 0.52	 0.47
4				 0.67	 0.32
5				 0.88	 0.05
6				 0.86	 0.04
7				 0.90	 0.04
8				 0.92	 0.07

Figure 5.11: Predictions of the decoder for the image set containing morphed images between the classes 'A' and 'N'.
Column 4 contains the reconstructed images for the unmasked squash vector of column 3. The images in column 5 and 6 were created by masking all squash vectors except for class 'A' or class 'N', respectively. The numbers below the images $||v_A||$ and $||v_N||$ are the lengths of the squash vector of the specific class.

lutional layer. The applied filters are comparatively large (9 pixels), and thus a small change of a feature could lead to a high change in the input of the low-level capsules. This could also be an explanation for the reason of the missing restoration of the letter's upper part in image 3 and 4. In this case, neither the features for class 'A' nor for class 'N' seem to be sufficiently found, so that the upper part of the letter is not predicted for any of the classes. This seems to result in a threshold for the detection of a class which is comparable to the human perception. A figure can be assigned to multiple classes which results in two comparatively high squash vectors. The masked predictions are restored properly while the unmasked prediction is a combination of both that contains the features of the unmasked predictions. This threshold could work as a support to investigate the features that are crucial for the capsule-decoder-network to detect a class. For class 'K' this is for instance demonstrated from image 4 to 5. In image 4, the arc of the 'R' is too wide to be counted for class 'K'. In image 5, the arc is more narrow and subsequently, the squash vector rises strongly. For class 'R' the threshold is visible from image 7 to image 8: In image 7 a small connection between the vertical line and the upwards slash exists and the class 'R' has a comparatively high squash vector.

For class 'O' in figure 5.10, the feature to recognize an 'O' is the completion of a circle with a sufficient intensity as shown from image 4 to image 5. The opposite seems to work as threshold for the detection of class 'C', whose squash vector length drops from image 5 to image 6. One important feature for class 'N' seems to be the height of its right line as shown in figure 5.11. The squash vector length drops from image 2 to image 3. The still comparatively high length drops a second time from image 5 to image 6, as soon as the inner triangle of the letter gets too large. For the class 'A', the squash vector length rises between image 2 to image 3, when the right line of the letter is small enough to be considered as residual. Remarkably the existence of the inverted V-shape seems to have a higher impact on the prediction of class 'A' that the horizontal line. The reason could be the existence of class 'B' whose upper part is similar to the upper part of class 'A'. As a consequence, the capsule-decoder-network relies more on the inverted V-shape.

The exploration of the threshold for different letters gives information about the important features for the capsule-decoder-network to recognize a class. Because the threshold for two letter is not on the same point, ambiguous images are represented by two medium sized squash vectors.

6 Summary of Results

The goal of this thesis was the evaluation of capsule networks for their capability to generate and explain keyword rankings. Therefore, a capsule-decoder-network was created in Python using Tensorflow and Keras functions. It was trained on 16384 images belonging to 12 classes of handwritten digits taken from the EMNIST dataset [17]. The architecture of the capsule-decoder-network was based on the architecture proposed in [16]. Because the reduced EMNIST dataset contains two additional classes to the MNIST dataset [20] used in [16], the number of high-level capsules was set to 12 in this model. Subsequently, the model's training parameters, which were also oriented on [16], were modified to generate the best results for this application: The number of routings inside the capsule layer was set to 5 instead of 3. The number of neurons in the first and second dense layer of the decoder was increased by a factor of four to handle the increased variation of handwritten letters compared to handwritten digits. The ratio of the capsule network's loss to the capsule-decoder-network's loss was set to $1 : 0.105$. To evaluate the model, the loss and accuracy of the capsule network, the loss of the capsule-decoder-network and the weighted sum of both losses was recorded.

The trained model generated squash vectors, that were used to create image rankings for each of the 12 classes of the EMNIST dataset. For each class, a dataset of 256 test images was created, that contained only letters of the considered class. Each class specific dataset was inserted into the trained model and the predicted label, the decoded image and the squash vector of each input image were stored. To verify the usage of the squash vector for a ranking, a distribution of the length of the present squash vectors was created. The majority of squash vectors for a class present in the image had a length between 0.70 and 0.99. A few samples were found that generated shorter squash vectors. Decoded images were created with the capsule-decoder-network based on the squash vectors for the present class. To support the approach to apply squash vectors for image rankings, the structural similarity was calculated between the decoded images of masked squash vectors and the original images. A correlation was found for longer squash vectors to account for decoded images with a higher SSIM to their original than shorter squash vectors. The creation of image rankings based on the squash vector was reasoned by this correlation. For each class-specific dataset of 256 images, a ranking was created for exemplary images that had a squash vector between 0.70 and 0.95. Using the masked squash

array, the area that was assigned to the present class by the capsule-decoder-network, was visualized. The decoded images from masked squash arrays appear smoother with less irregularities and with less details than the corresponding unmasked prediction of the input image. In the decoded images from the unmasked squash arrays more irregularities appear which are either an enhancement or a downsizing of features. In some cases, details that are missing in the decoded image of the masked squash array are found in the image of the unmasked squash array. However, the restoration of details is unreliable because no correlation between the SSIM of the decoded image from the unmasked squash vector and the original was found.

To explain the irregularities in decoded images based on unmasked squash arrays, decoded images were created based on squash vectors of classes that were not present in an image but simultaneous had a similar high length. This approach was performed for three sets of the classes 'A', 'C' and 'K'. Each set consisted of 12 images that generated differently large squash vectors for the present class. For each set, decoded images were created for the present squash vector and for the one or two largest non-present squash vectors. The decoded images of non-present squash vectors explained the reason for the irregularities in decoded images of unmasked squash arrays: A superposition of each decoded image from single squash vectors takes place in the decoded image based on the unmasked squash array. The impact of each squash vector on the decoded image depends on its size: The larger a squash vector, the higher is the intensity of the restored letter in the decoded image of the unmasked squash array. This behaviour leads to the conclusion, that the squash vectors of non-present classes contain those features that the capsule-decoder-network recognized as part of this non-present class. Through the length of the non-present squash vectors the certainty of the capsule-decoder-network for the class is calculable.

To examine the scale of non-present features for all images in each class-specific dataset, the length of the largest squash vector was examined. The length and the class of this squash vector were stored. This led to an overview matrix for all classes, in which features of non-present classes were mostly recognized within their images. Through the lengths comparison of the present and the non-present squash vector, a relation of the certainty for the present class to the certainty for the non-present class was assessable. It was found that in a squash array with one large squash vectors no second large squash

vector was generated. When the squash vector's length of the present class decreased, the number of occurrences of a larger non-present squash vector increased. This is explainable through a possible ambiguity of features: Features that generate high squash vectors are clearly assigned to one class. Features, that are less unique, have a higher chance to be assigned to more than one class. A large squash vector does not suppress the assignment of features to other classes, but the capsule-decoder-network is trained to assign these features only to one specific class.

Based on the overview matrix, the three frequent combinations for the recognition of non-present features in a specific class were examined. These combinations were the letters 'K' and 'R', 'C' and 'O', as well as 'A' and 'N'. The similarity perceived by the capsule-decoder-network in these classes was used to determine the features important for the detection through the capsule-decoder-network. Therefore, a set of 8 images was created for each combination. These sets contained manually, artificially created images whose letters were gradually transformed from one class to the other. The images were predicted by the model and decoded images were created based on the unmasked squash array and on both squash vectors for the original classes. The length of the squash vector increased or decreased with the construction or deconstruction of the corresponding class. A threshold was found for each class in the sets by which the length of the squash vector increased or dropped by a value of ≥ 0.3. Through this threshold specific features for the recognition of a class were found, like the length of the right line of the letter 'N' or the connection between the upper and lower part of a 'C' to recognize the class 'O'. The reason for this threshold was in the 9×9 convolutional filters that prepare the input to the low-level capsules. These filters concatenate a comparatively large area and use the leaky-ReLU [32] function as activation resulting in a possible large increase of values in low-level capsules by a slight modification of a feature.

In the transformation from one class to another, the threshold for both classes was found in different images. Therefore, the capsule-decoder-network is capable of detecting ambiguous images by the length of the squash vector and simultaneously explaining the reasons for this decision by the decoded images of the corresponding classes.

Conclusively, it was shown that the length of the squash vector is a quantifiable measurement, which is directly built into the capsule-decoder-network, to evaluate the affiliation of an image to a specific class. Beside its length, the perceived features are restorable by a

decoder. Further considerations about the squash vectors's explanatory potential and its possible usage for search engine rankings are carried out in the subsequent section 6.1.

6.1 Discussion of the Results as Explanatory Approach

In the introduction of this thesis, the term 'explainability' for an AI system was described as '*ability to explain technical process*' so that they are '*understood by human beings*'. [8] The goal was the explanation of an image ranking based on the perception of the capsule-decoder-network. Image ranking were defined as categorization of images based on their affiliation to a specific class.

The length of the squash vector provided a quantifiable possibility to rank images of one class based on the certainty of the capsule-decoder-network. The features, that led to this decision, are restorable by the decoder. These features relate to the areas, that contributed to the increase of the certainty for the class. Through the restoration of the features, the decision of the capsule-decoder-network becomes comprehensible. It is possible to estimate the scope of the incorrect prediction by the length of the squash vector's class and explain the reasons for this decision by the letter fragments in the decoded image. These letter fragments are mostly human-understandable. Furthermore, the features used to detect a class, are also similar to the features that a human would use for the detection.

The aforementioned explanatory approaches for CNN are post-hoc approaches that create their visualization and explanation after the training. However, the squash vector is directly integrated into the capsule-decoder-network and it is directly used to predict the class and to decode the image. This close connection between the prediction and the visualization indicates the validity of the resulting explanatory approach based on the value of the squash vector and the decoded image. Through this connection, the decoding of the squash vector provides a closer look to the perceived features than common approaches as Grad-CAM [13] or the creation of saliency maps. [14, 21, 22] Using the decoder, one could argue that the capsule-decoder-network does not actually use these features. However, the fact that the restoration is still possible after two convolutional layers and a capsule layer, indicates that the downsized, encoded features are stored in the high-level capsules. Thereby, the decoding is a reconstruction of existing features into a human-understandable format which is a large advantage in comparison to CNN.

The decoded images and the length of the squash vector are a valid base to create im-

age rankings and to explore the features that are used by the capsule-decoder-network. The restored features are human-comprehensible and can especially support the detection and explanation of ambiguities in images.

Despite the advantages of capsule networks regarding the explainability, this approach does not lead to a complete explanation of the capsule-decoder-network's decision. The squash vector is a tool that generates a comprehensible explanation, but it does not provide the reasons for the decision of the capsule-decoder-network: Despite the decoded letter fragments being understandable, it remains unclear, why they were interpreted as a specific non-present class. The reason therefore lays probably in the convolutional layers, that modify the features for the capsules.

An obstacle in the usage of capsule networks is their comparably time-consuming and error-prone implementation process. The dynamic routing algorithm generates additional parameters for which the adjustment experience is limited. Besides that it offers more possibilities, the implementation and optimization is more elaborate.

All summarized, the explanation of image rankings is in fact possible by using capsule networks. The explanation works on the vision of the capsule-decoder-network and the results are comparatively human-understandable. A large advantage is the simultaneous evaluation of all classes. Especially in search engines, this simultaneous evaluation could support the findings for alternative keywords. Therefore, capsule networks have the potential to be applied in search engines in the future.

6.2 Further Research

In this thesis, the capsule-decoder-network was trained on a comparatively simple dataset. For further research, the application of capsule networks for more complex datasets could be evaluated. For this case, modifications will be necessary because [16] already described difficulties in the training to the CIFAR [46] dataset. In this thesis, an attempt to train on the intel dataset also failed. To explore the potential of capsule networks more extensively, an implementation of capsule layers in the common libraries as Tensorflow or Keras would be especially helpful. By making capsule networks more accessible the research on this topic is encouraged and the potential of capsule networks can be investigated to a greater degree. Thereby, a special look could be taken to the impact of the decoder on the restored images and on the coupling coefficients, because

their values could evaluate the impact of the dynamic routing to the learning process of the capsule-decoder-network.

A Distribution of Squash Vectors for class specific datasets 'B' to 'Z'

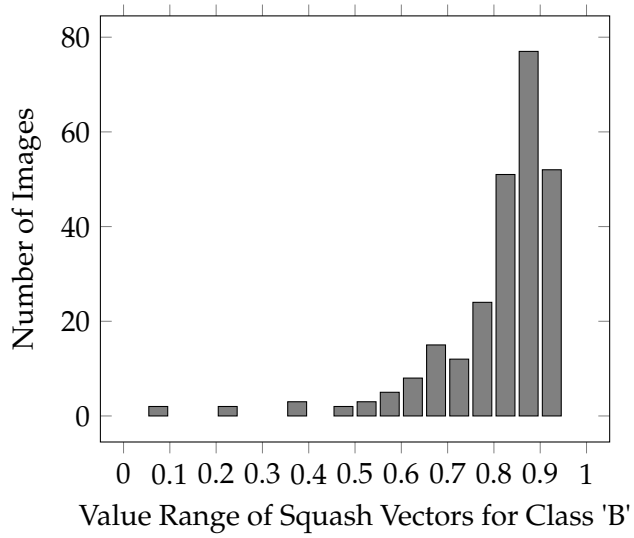


Figure A.1: Distribution of squash vectors from 256 test images containing class 'B'

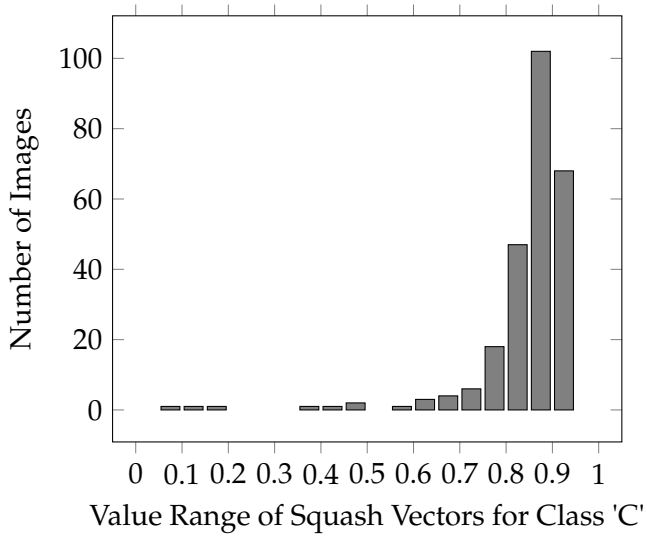


Figure A.2: Distribution of squash vectors from 256 test images containing class 'C'

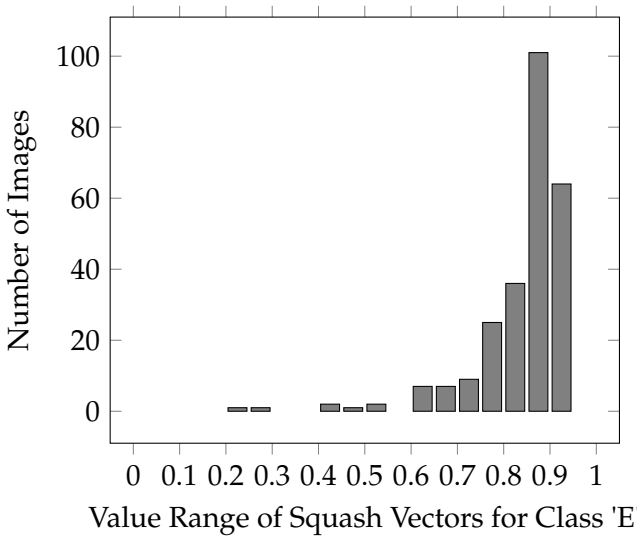


Figure A.3: Distribution of squash vectors from 256 test images containing class 'E'

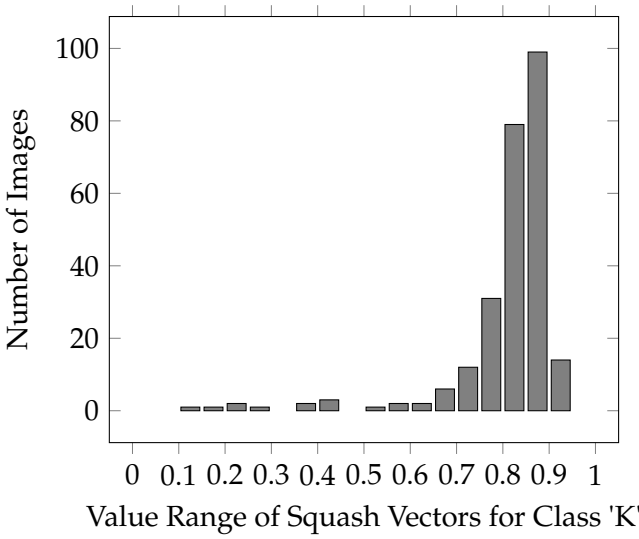


Figure A.4: Distribution of squash vectors from 256 test images containing class 'K'

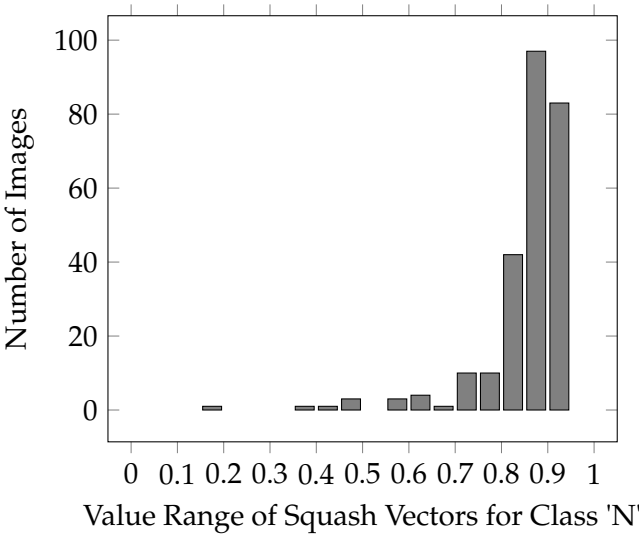


Figure A.5: Distribution of squash vectors from 256 test images containing class 'N'

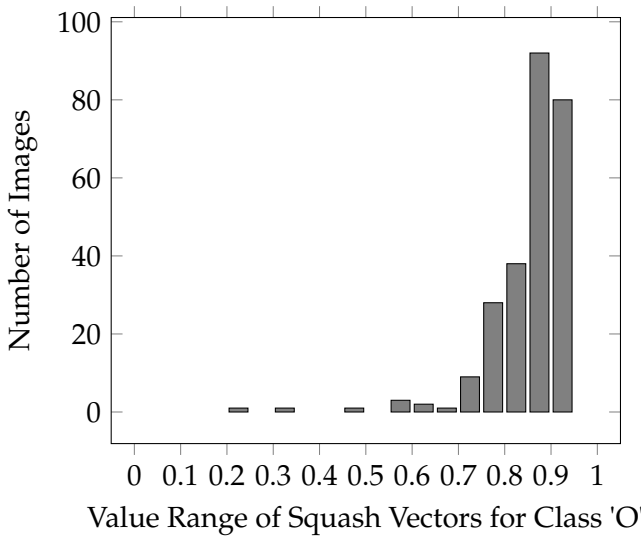


Figure A.6: Distribution of squash vectors from 256 test images containing class 'O'

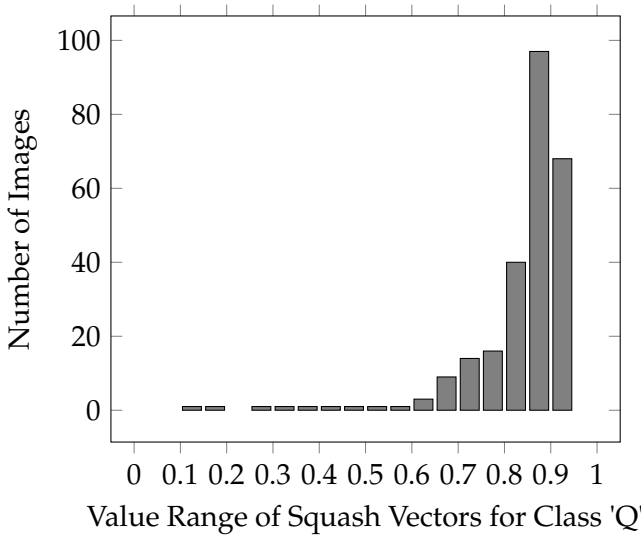


Figure A.7: Distribution of squash vectors from 256 test images containing class 'Q'

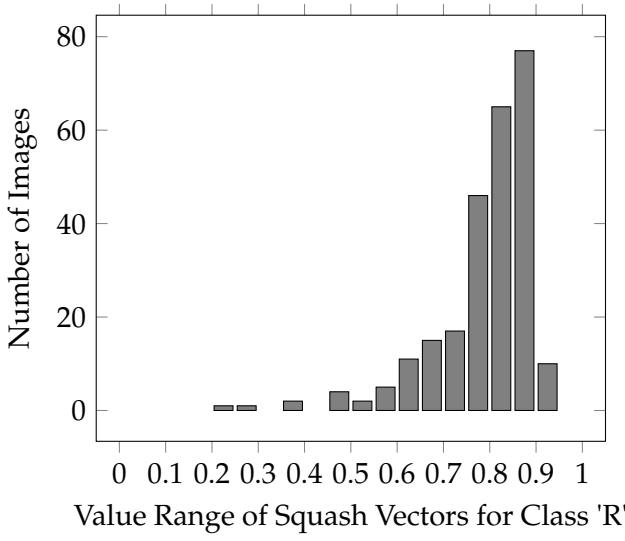


Figure A.8: Distribution of squash vectors from 256 test images containing class 'R'

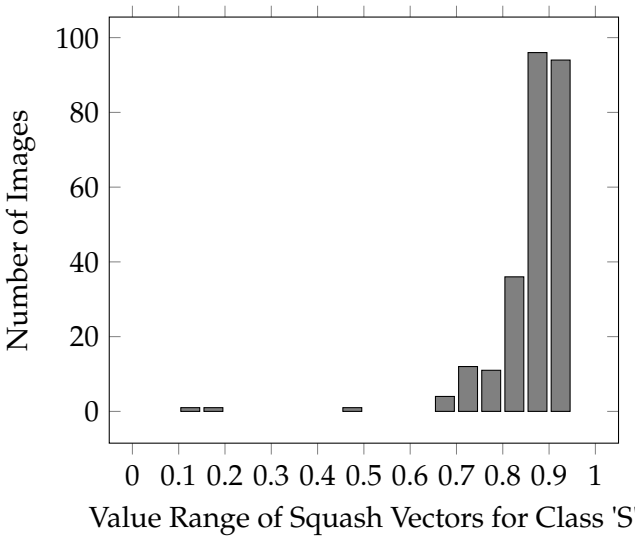


Figure A.9: Distribution of squash vectors from 256 test images containing class 'S'

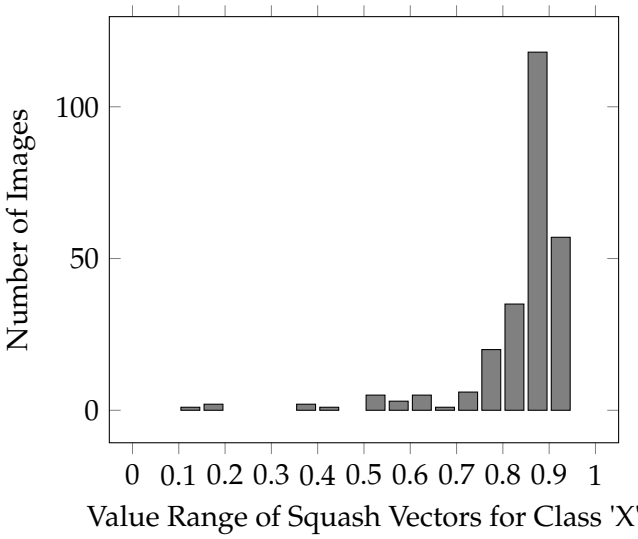


Figure A.10: Distribution of squash vectors from 256 test images containing class 'X'

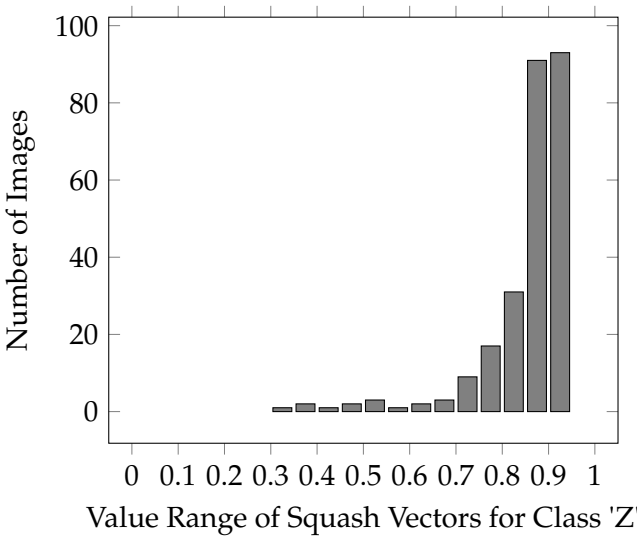


Figure A.11: Distribution of squash vectors from 256 test images containing class 'Z'

B Structural Similarity for Decoded Images of Masked Squash Vectors

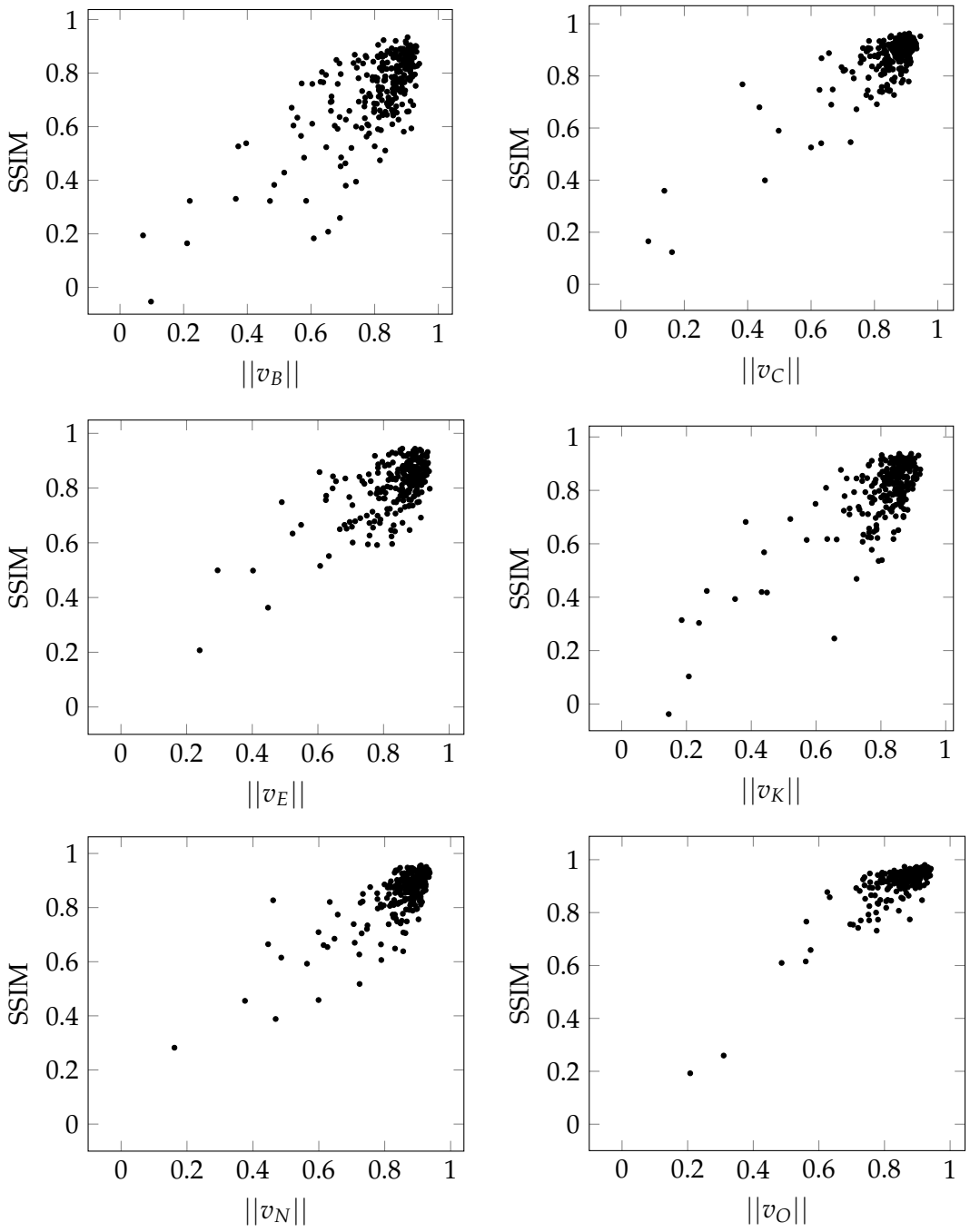


Figure B.1: Plots of structural similarity (SSIM) for the squash vectors of class 'B' to 'O'

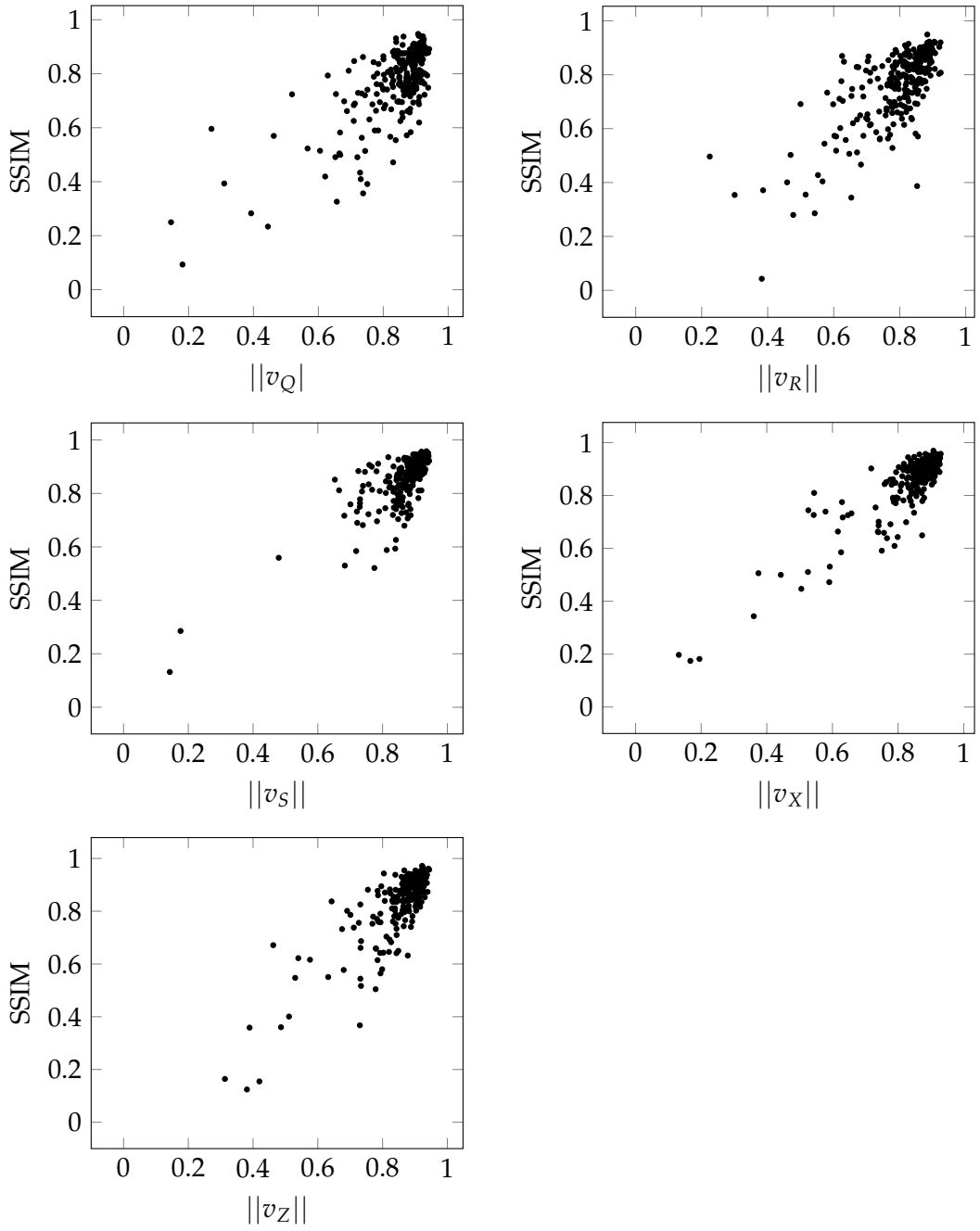


Figure B.2: Plots of structural similarity (SSIM) for the squash vectors of class 'S' to 'Z'

C Rankings for Masked and Unmasked Squash Vectors 'B' - 'Z'

	Input Image									
Original	-	-	-	-	-					
Unmasked Prediction	-	-	-	-	-					
Masked Prediction	-	-	-	-	-					
$\|\|v_B\|\|$	0.99	0.98	0.97	0.96	0.95	0.94	0.93	0.92	0.91	0.90
Original										
Unmasked Prediction										
Masked Prediction										
$\|\|v_B\|\|$	0.89	0.88	0.87	0.86	0.85	0.84	0.83	0.82	0.81	0.80
Original										-
Unmasked Prediction										-
Masked Prediction										-
$\|\|v_B\|\|$	0.79	0.78	0.77	0.76	0.75	0.74	0.73	0.72	0.71	0.70

Figure C.1: Ranking of images for class 'B' based on the length of the corresponding squash array for three intervals between 0.7 and 0.99.

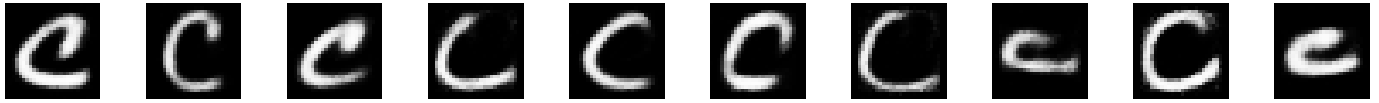
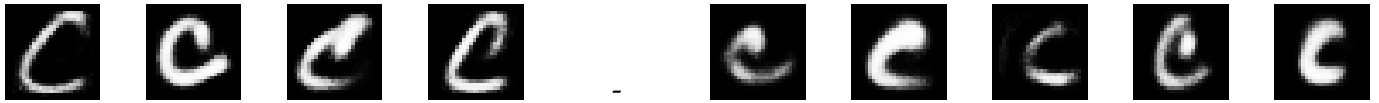
	Input Image									
Original	-	-	-	-	-					
Unmasked Prediction	-	-	-	-	-					
Masked Prediction	-	-	-	-	-					
$\|\|v_C\|\|$	0.99	0.98	0.97	0.96	0.95	0.94	0.93	0.92	0.91	0.90
Original										
Unmasked Prediction										
Masked Prediction										
$\|\|v_C\|\|$	0.89	0.88	0.87	0.86	0.85	0.84	0.83	0.82	0.81	0.80
Original					-					
Unmasked Prediction					-					
Masked Prediction					-					
$\|\|v_C\|\|$	0.79	0.78	0.77	0.76	0.75	0.74	0.73	0.72	0.71	0.70

Figure C.2: Ranking of images for class 'C' based on the length of the corresponding squash array for three intervals between 0.7 and 0.99.

	Input Image									
Original	-	-	-	-	-					
Unmasked Prediction	-	-	-	-	-					
Masked Prediction	-	-	-	-	-					
$\|\|v_E\|\|$	0.99	0.98	0.97	0.96	0.95	0.94	0.93	0.92	0.91	0.90
Original										
Unmasked Prediction										
Masked Prediction										
$\|\|v_E\|\|$	0.89	0.88	0.87	0.86	0.85	0.84	0.83	0.82	0.81	0.80
Original								-		
Unmasked Prediction								-		
Masked Prediction								-		
$\|\|v_E\|\|$	0.79	0.78	0.77	0.76	0.75	0.74	0.73	0.72	0.71	0.70

Figure C.3: Ranking of images for class 'E' based on the length of the corresponding squash array for three intervals between 0.7 and 0.99.

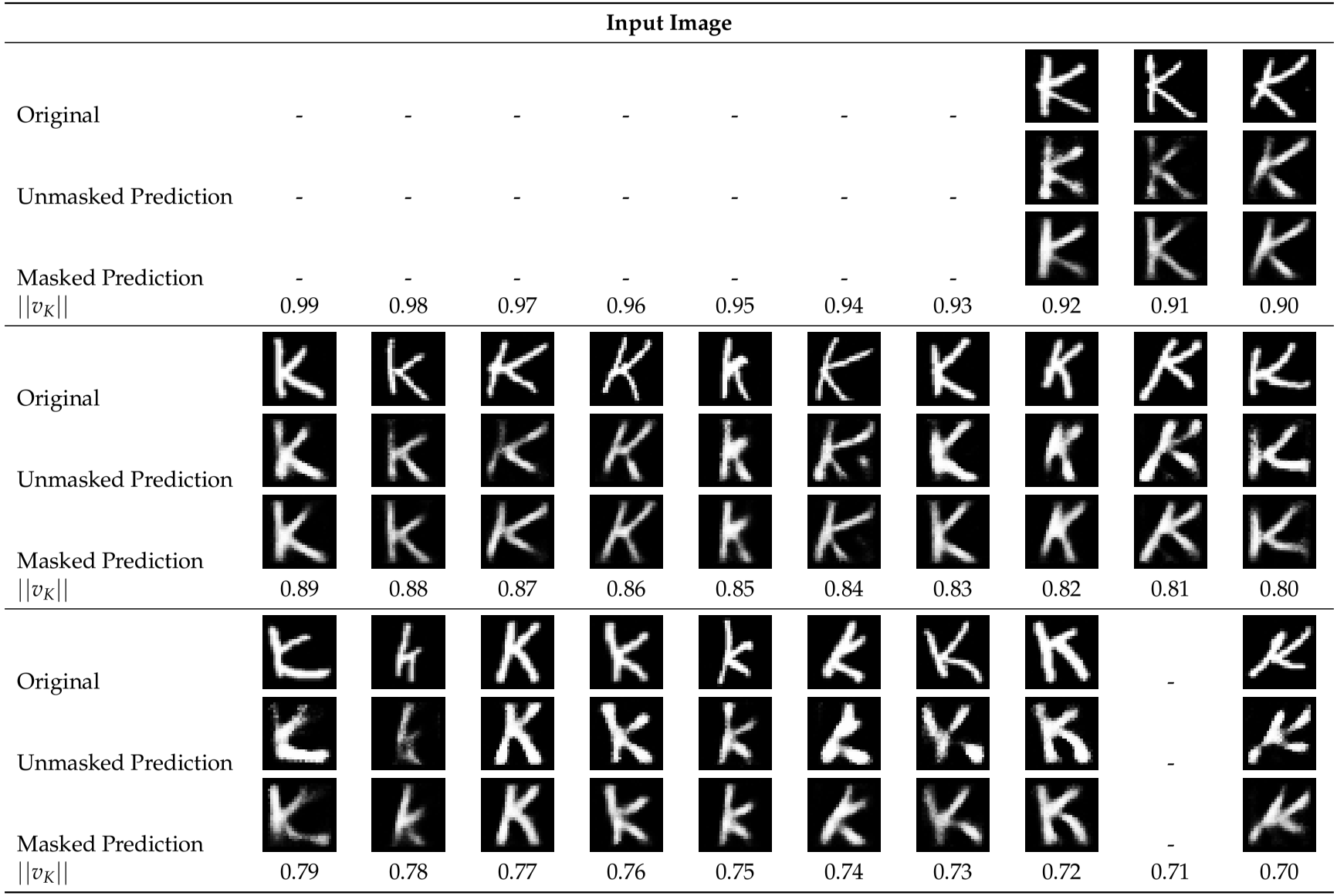


Figure C.4: Ranking of images for class 'A' based on the length of the corresponding squash array for three intervals between 0.7 and 0.99.

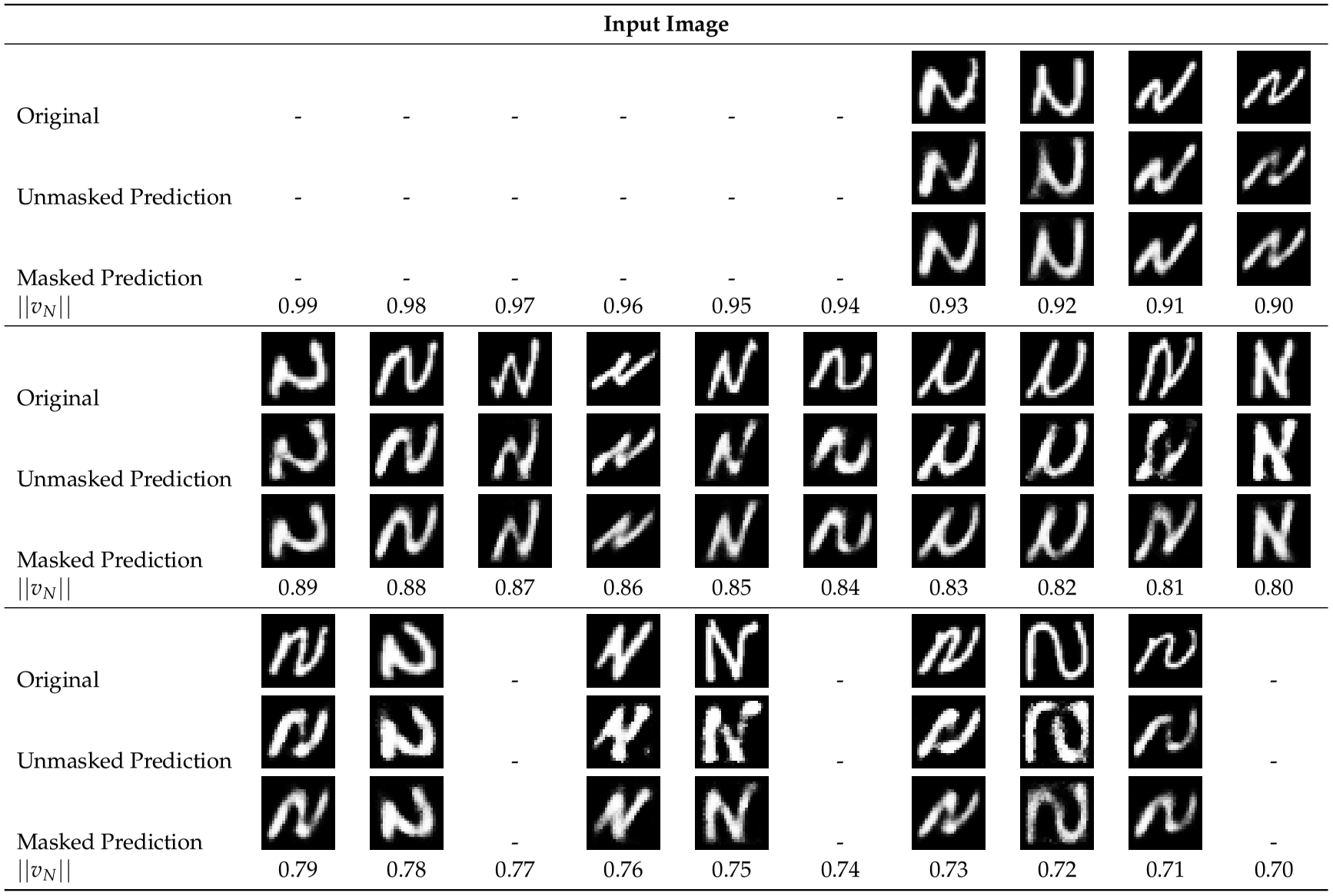


Figure C.5: Ranking of images for class 'N' based on the length of the corresponding squash array for three intervals between 0.7 and 0.99.

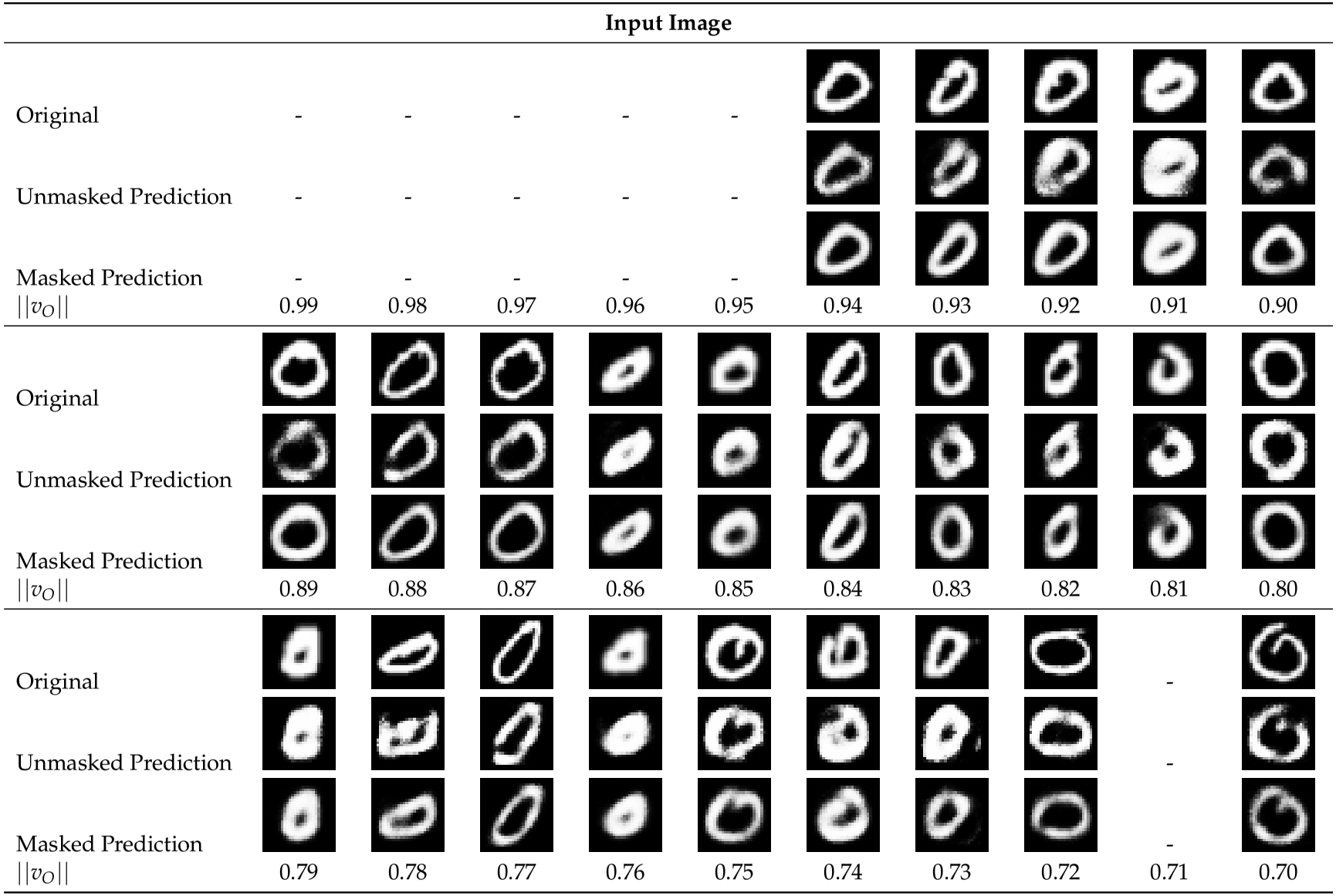


Figure C.6: Ranking of images for class 'O' based on the length of the corresponding squash array for three intervals between 0.7 and 0.99.

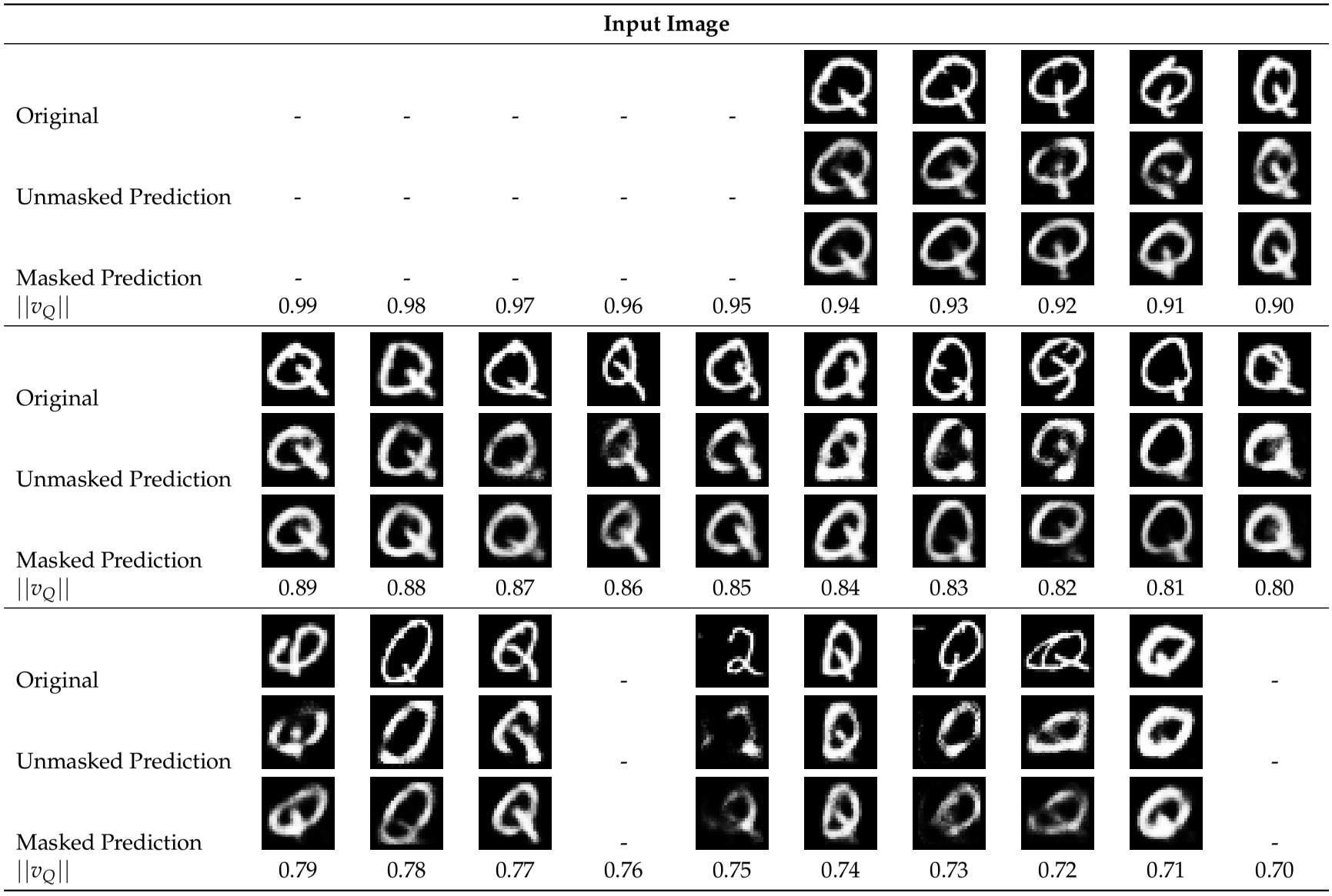


Figure C.7: Ranking of images for class 'Q' based on the length of the corresponding squash array for three intervals between 0.7 and 0.99.

	Input Image									
Original	-	-	-	-	-	-	-			
Unmasked Prediction	-	-	-	-	-	-	-			
Masked Prediction	-	-	-	-	-	-	-			
$\|\|v_R\|\|$	0.99	0.98	0.97	0.96	0.95	0.94	0.93	0.92	0.91	0.90
Original										
Unmasked Prediction										
Masked Prediction										
$\|\|v_R\|\|$	0.89	0.88	0.87	0.86	0.85	0.84	0.83	0.82	0.81	0.80
Original										
Unmasked Prediction										
Masked Prediction										
$\|\|v_R\|\|$	0.79	0.78	0.77	0.76	0.75	0.74	0.73	0.72	0.71	0.70

Figure C.8: Ranking of images for class 'R' based on the length of the corresponding squash array for three intervals between 0.7 and 0.99.

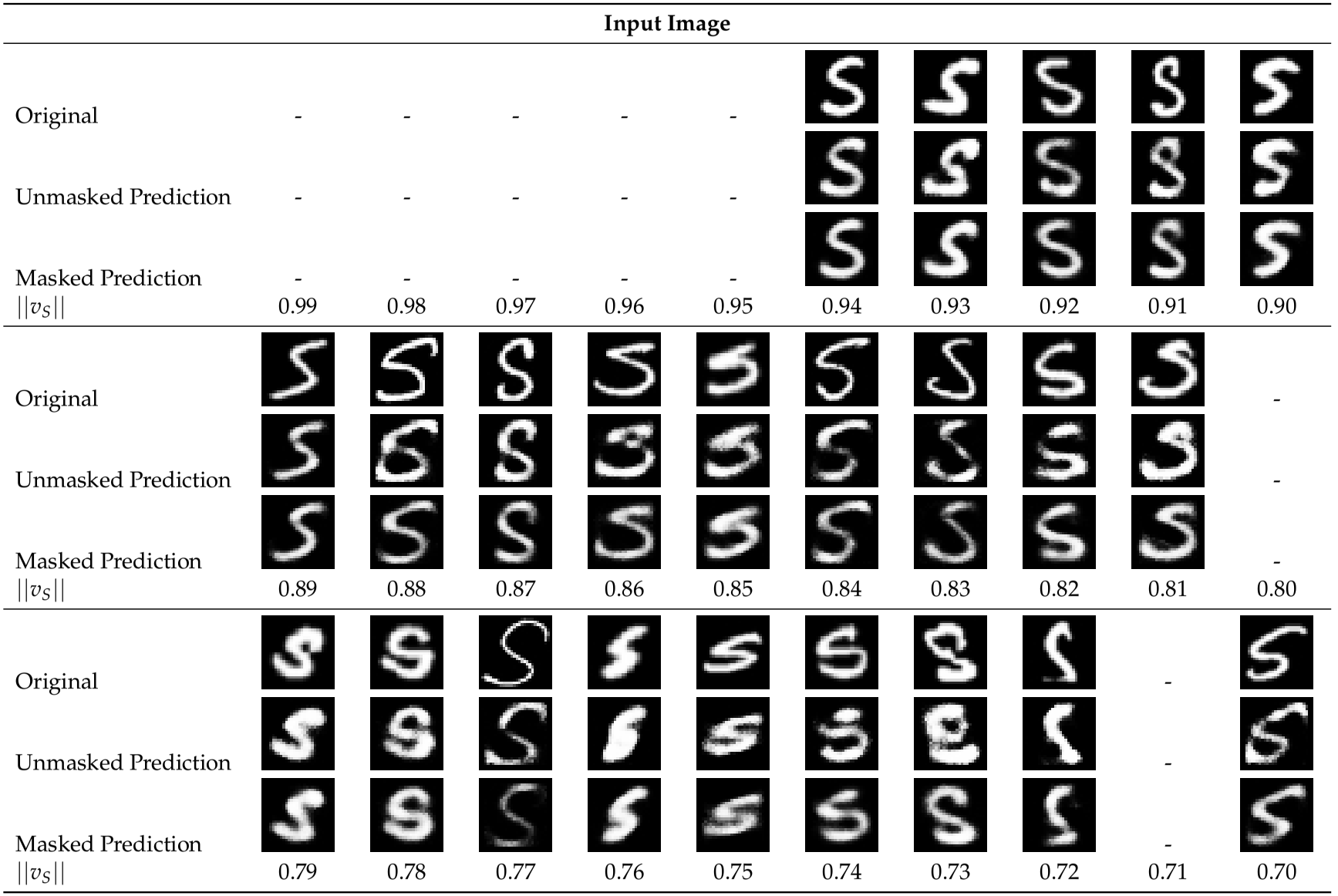


Figure C.9: Ranking of images for class 'S' based on the length of the corresponding squash array for three intervals between 0.7 and 0.99.

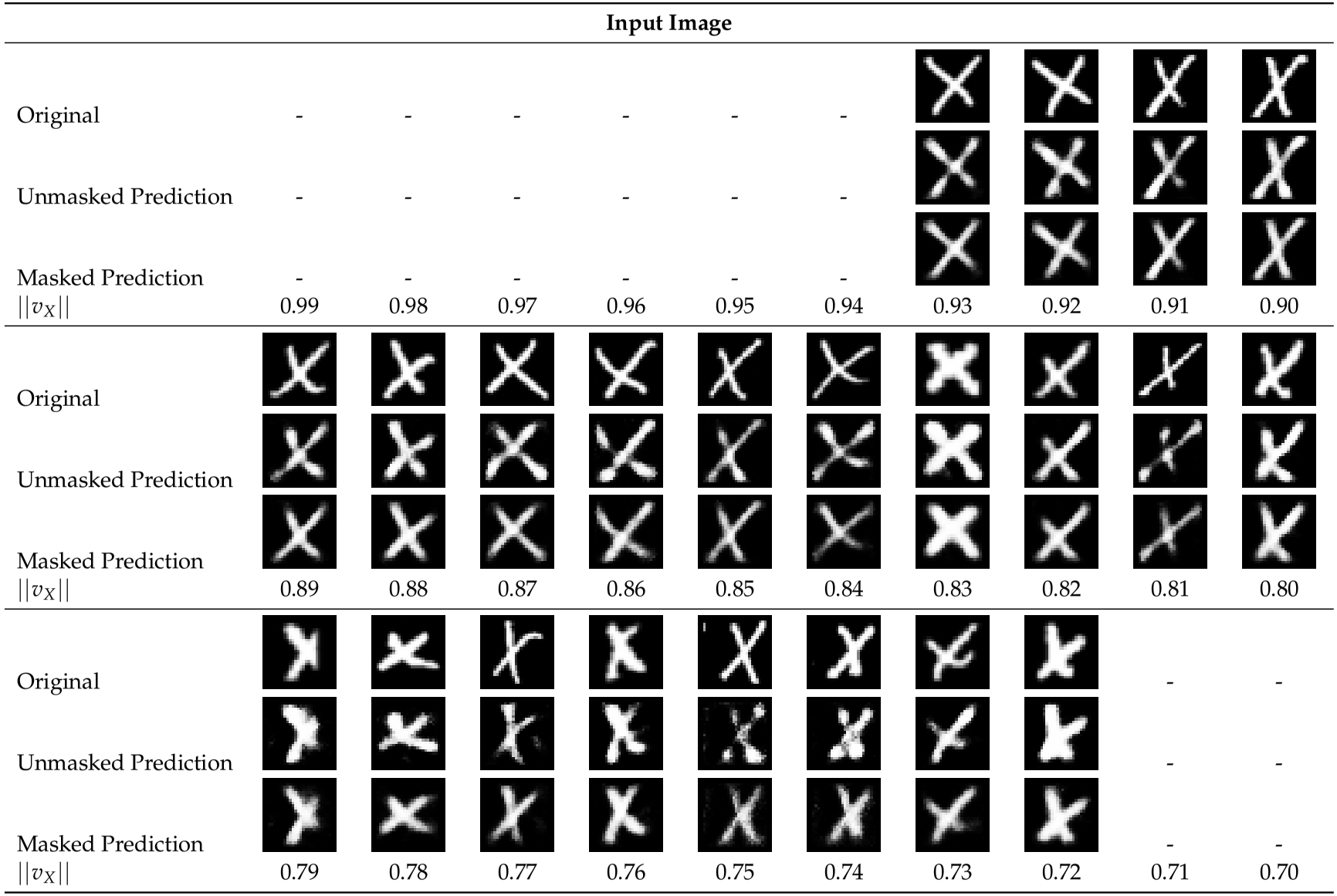


Figure C.10: Ranking of images for class 'X' based on the length of the corresponding squash array for three intervals between 0.7 and 0.99.

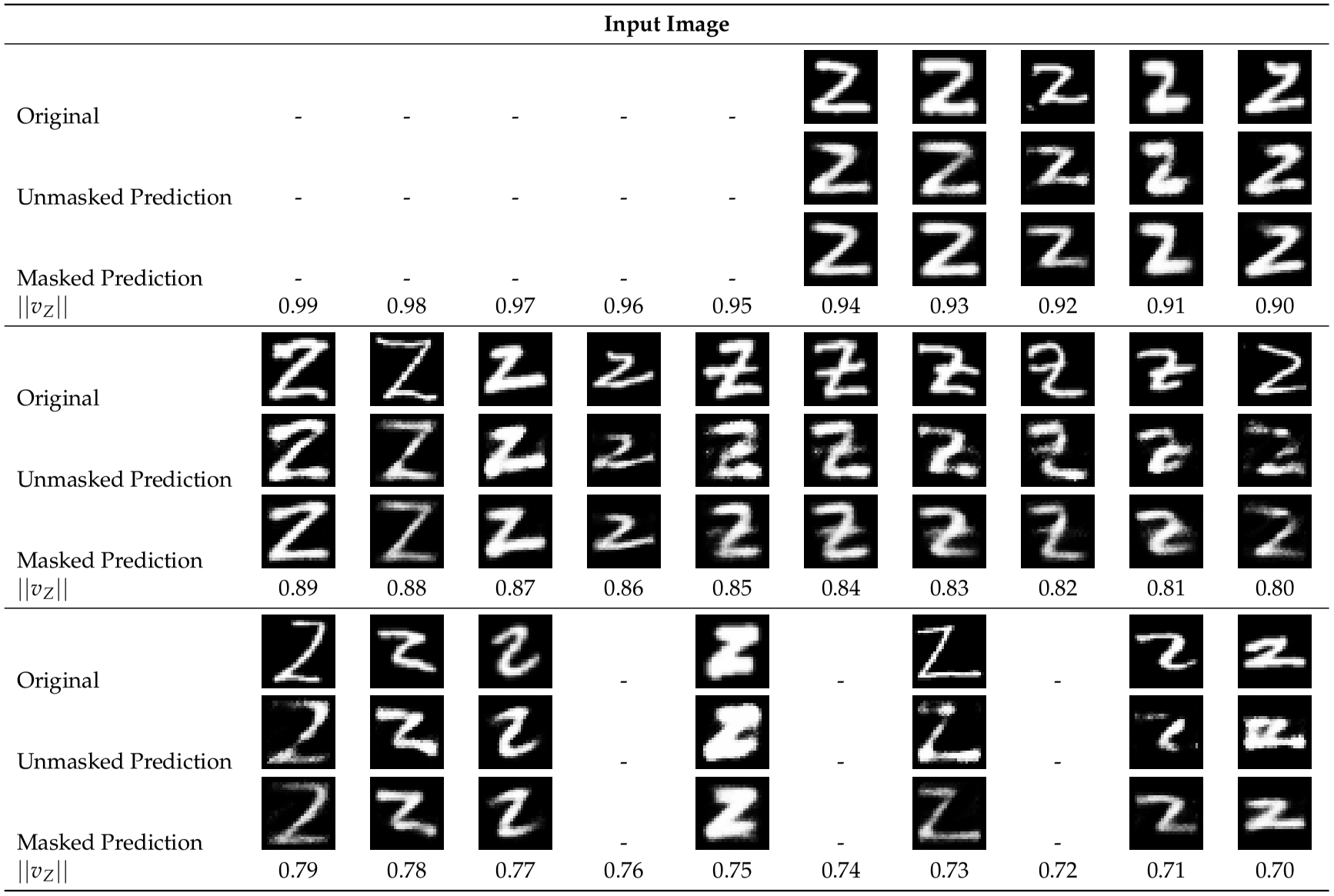


Figure C.11: Ranking of images for class 'Z' based on the length of the corresponding squash array for three intervals between 0.7 and 0.99.

D Comparison of Largest Non-Present Squash Vectors

Table D.1: Overview of the squash vectors lengths for the remaining classes in comparison to the length of the squash vector for class 'B'.

Squash Vector 'B' Length Interval	**Largest Squash Vector of Non-Present Class**										
	A	C	E	K	N	O	Q	R	S	X	Z
	Max. length of largest squash vector of non-present class Min. length of largest squash vector of non-present class										
$0.9 \leq \|\|v_A\|\| < 1.0$	18	5	12	1	0	2	1	2	1	4	6
	0.35	0.26	0.25	0.11	-	0.18	0.09	0.11	0.16	0.35	0.14
	0.08	0.09	0.08	0.11	-	0.17	0.09	0.09	0.16	0.14	0.10
$0.8 \leq \|\|v_A\|\| < 0.9$	17	14	21	7	1	11	5	20	16	8	8
	0.46	0.31	0.37	0.44	0.15	0.38	0.24	0.40	0.27	0.31	0.52
	0.13	0.11	0.10	0.14	0.15	0.12	0.13	0.10	0.11	0.10	0.11
$0.7 \leq \|\|v_A\|\| < 0.8$	7	4	4	0	0	5	2	9	3	0	2
	0.35	0.30	0.51	-	-	0.54	0.46	0.50	0.29	-	0.60
	0.20	0.21	0.21	-	-	0.22	0.22	0.21	0.23	-	0.41
$0.6 \leq \|\|v_A\|\| < 0.7$	2	1	4	1	0	9	2	2	1	0	1
	0.56	0.19	0.28	0.41	-	0.61	0.35	0.34	0.27	-	0.24
	0.31	0.19	0.24	0.41	-	0.30	0.34	0.25	0.27	-	0.24
$0.5 \leq \|\|v_A\|\| < 0.6$	1	1	2	1	0	1	0	2	0	0	0
	0.53	0.30	0.65	0.29	-	0.66	-	0.51	-	-	-
	0.53	0.30	0.49	0.29	-	0.66	-	0.42	-	-	-
$0.4 \leq \|\|v_A\|\| < 0.5$	0	0	0	0	0	1	0	0	0	0	1
	-	-	-	-	-	0.56	-	-	-	-	0.69
	-	-	-	-	-	0.56	-	-	-	-	0.69
$0.3 \leq \|\|v_A\|\| < 0.4$	0	0	1	0	0	0	0	2	0	0	0
	-	-	0.31	-	-	-	-	0.79	-	-	-
	-	-	0.31	-	-	-	-	0.66	-	-	-
$0.2 \leq \|\|v_A\|\| < 0.3$	0	0	0	1	0	1	0	0	0	0	0
	-	-	-	0.31	-	0.68	-	-	-	-	-
	-	-	-	0.31	-	0.68	-	-	-	-	-
$0.1 \leq \|\|v_A\|\| < 0.2$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
$0.0 \leq \|\|v_A\|\| < 0.1$	0	1	0	0	0	0	1	0	0	0	0
	-	0.55	-	-	-	-	0.83	-	-	-	-
	-	0.55	-	-	-	-	0.83	-	-	-	-
Sum of Instances	45	26	44	11	1	30	11	37	21	12	18

Table D.2: Overview of the squash vectors lengths for the remaining classes in comparison to the length of the squash vector for class 'C'.

Squash Vector 'C' Length Interval	**Largest Squash Vector of Non-Present Class**										
	A	C	E	K	N	O	Q	R	S	X	Z
	Max. length of largest squash vector of non-present class Min. length of largest squash vector of non-present class										
$0.9 \leq \|\|v_A\|\| < 1.0$	1	2	11	7	1	20	5	11	3	1	6
	0.11	0.09	0.15	0.17	0.12	0.15	0.18	0.24	0.13	0.12	0.13
	0.11	0.08	0.10	0.08	0.12	0.08	0.12	0.08	0.06	0.12	0.10
$0.8 \leq \|\|v_A\|\| < 0.9$	2	11	21	17	3	35	16	13	10	3	18
	0.2	0.41	0.33	0.26	0.19	0.28	0.29	0.38	0.39	0.15	0.26
	0.16	0.12	0.08	0.09	0.13	0.08	0.10	0.12	0.07	0.07	0.09
$0.7 \leq \|\|v_A\|\| < 0.8$	0	1	9	1	0	4	2	3	1	0	3
	-	0.32	0.43	0.18	-	0.26	0.30	0.32	0.15	-	0.32
	-	0.32	0.21	0.18	-	0.22	0.27	0.25	0.15	-	0.32
$0.6 \leq \|\|v_A\|\| < 0.7$	3	0	1	1	0	2	0	0	0	0	0
	0.30	-	0.36	0.21	-	0.55	-	-	-	-	-
	0.22	-	0.36	0.21	-	0.48	-	-	-	-	-
$0.5 \leq \|\|v_A\|\| < 0.6$	0	0	0	0	0	0	1	0	0	0	0
	-	-	-	-	-	-	0.26	-	-	-	-
	-	-	-	-	-	-	0.26	-	-	-	-
$0.4 \leq \|\|v_A\|\| < 0.5$	0	0	0	0	0	1	1	1	0	0	0
	-	-	-	-	-	0.77	0.46	0.38	-	-	-
	-	-	-	-	-	0.77	0.46	0.38	-	-	-
$0.3 \leq \|\|v_A\|\| < 0.4$	0	0	0	0	0	0	0	1	0	0	0
	-	-	-	-	-	-	-	0.67	-	-	-
	-	-	-	-	-	-	-	0.67	-	-	-
$0.2 \leq \|\|v_A\|\| < 0.3$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
$0.1 \leq \|\|v_A\|\| < 0.2$	0	0	1	0	0	0	0	1	0	0	0
	-	-	0.6	-	-	-	-	0.71	-	-	-
	-	-	0.6	-	-	-	-	0.71	-	-	-
$0.0 \leq \|\|v_A\|\| < 0.1$	0	0	0	0	0	0	0	0	1	0	0
	-	-	-	-	-	-	-	-	0.50	-	-
	-	-	-	-	-	-	-	-	0.50	-	-
Sum of Instances	6	14	43	26	4	62	25	30	15	4	27

Table D.3: Overview of the squash vectors lengths for the remaining classes in comparison to the length of the squash vector for class 'E'.

Squash Vector 'E' Length Interval	**Number of Classes with Second Largest Squash Vector**										
	A	B	C	K	N	O	Q	R	S	X	Z
	Max. length of largest squash vector of non-present class Min. length of largest squash vector of non-present class										
$0.9 \leq \|\|v_A\|\| < 1.0$	3	8	5	14	4	3	5	4	10	0	7
	0.14	0.18	0.24	0.23	0.21	0.17	0.19	0.20	0.20	-	0.23
	0.08	0.07	0.10	0.07	0.08	0.12	0.07	0.12	0.10	-	0.10
$0.8 \leq \|\|v_A\|\| < 0.9$	3	17	14	19	3	5	9	33	19	7	9
	0.15	0.39	0.30	0.40	0.21	0.16	0.38	0.63	0.34	0.26	0.35
	0.13	0.09	0.09	0.10	0.10	0.09	0.10	0.10	0.09	0.10	0.11
$0.7 \leq \|\|v_A\|\| < 0.8$	2	3	6	8	0	2	1	5	6	0	1
	0.38	0.27	0.35	0.53	-	0.23	0.22	0.55	0.28	-	0.3
	0.21	0.13	0.22	0.25	-	0.18	0.22	0.24	0.22	-	0.3
$0.6 \leq \|\|v_A\|\| < 0.7$	0	1	3	1	1	1	0	1	1	1	4
	-	0.54	0.56	0.37	0.31	0.21	-	0.47	0.46	0.24	0.36
	-	0.54	0.56	0.37	0.31	0.21	-	0.47	0.46	0.24	0.36
$0.5 \leq \|\|v_A\|\| < 0.6$	0	0	0	0	0	0	0	1	1	0	0
	-	-	-	-	-	-	-	0.41	0.42	-	-
	-	-	-	-	-	-	-	0.41	0.42	-	-
$0.4 \leq \|\|v_A\|\| < 0.5$	0	1	0	0	1	0	0	0	0	0	1
	-	0.24	-	-	0.66	-	-	-	-	-	0.47
	-	0.24	-	-	0.66	-	-	-	-	-	0.47
$0.3 \leq \|\|v_A\|\| < 0.4$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
$0.2 \leq \|\|v_A\|\| < 0.3$	0	0	0	1	0	0	0	0	0	0	1
	-	-	-	0.52	-	-	-	-	-	-	0.45
	-	-	-	0.52	-	-	-	-	-	-	0.45
$0.1 \leq \|\|v_A\|\| < 0.2$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
$0.0 \leq \|\|v_A\|\| < 0.1$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
Sum of Instances	8	30	28	43	9	11	15	44	37	8	23

Table D.4: Overview of the squash vectors lengths for the remaining classes in comparison to the length of the squash vector for class 'K'.

Squash Vector 'K' Length Interval	**Largest Squash Vector of Non-Present Class**										
	A	B	C	E	N	O	Q	R	S	X	Z
	Max. length of largest squash vector of non-present class Min. length of largest squash vector of non-present class										
$0.9 \leq \|\|v_A\|\| < 1.0$	0	1	3	2	1	0	1	2	0	3	1
	-	0.24	0.19	0.16	0.12	-	0.11	0.24	-	0.18	0.10
	-	0.24	0.11	0.15	0.12	-	0.11	0.12	-	0.12	0.10
$0.8 \leq \|\|v_A\|\| < 0.9$	4	4	9	14	14	0	3	74	3	47	6
	0.19	0.16	0.63	0.60	0.38	-	0.20	0.46	0.24	0.39	0.21
	0.11	0.10	0.12	0.12	0.12	-	0.13	0.09	0.13	0.08	0.07
$0.7 \leq \|\|v_A\|\| < 0.8$	3	1	6	2	3	0	3	10	0	15	0
	032	0.32	0.62	0.44	0.28	-	0.58	0.67	-	0.67	-
	0.15	0.32	0.19	0.28	0.26	-	0.18	0.14	-	0.14	-
$0.6 \leq \|\|v_A\|\| < 0.7$	1	0	0	0	0	0	0	2	0	5	0
	0.26	-	-	-	-	-	-	0.52	-	0.50	-
	0.26	-	-	-	-	-	-	0.31	-	0.31	-
$0.5 \leq \|\|v_A\|\| < 0.6$	1	0	0	0	0	0	0	1	0	1	0
	0.24	-	-	-	-	-	-	0.67	-	0.52	-
	0.24	-	-	-	-	-	-	0.67	-	0.52	-
$0.4 \leq \|\|v_A\|\| < 0.5$	0	0	1	0	0	0	0	1	0	0	1
	-	-	0.72	-	-	-	-	0.65	-	-	0.58
	-	-	0.72	-	-	-	-	0.65	-	-	0.58
$0.3 \leq \|\|v_A\|\| < 0.4$	0	0	1	0	0	0	0	0	0	1	0
	-	-	0.38	-	-	-	-	-	-	0.62	-
	-	-	0.38	-	-	-	-	-	-	0.62	-
$0.2 \leq \|\|v_A\|\| < 0.3$	2	0	0	1	0	0	0	0	0	0	0
	0.59	-	-	0.42	-	-	-	-	-	-	-
	0.44	-	-	0.42	-	-	-	-	-	-	-
$0.1 \leq \|\|v_A\|\| < 0.2$	1	0	1	0	0	0	0	0	0	0	0
	0.35	-	0.82	-	-	-	-	-	-	-	-
	0.35	-	0.82	-	-	-	-	-	-	-	-
$0.0 \leq \|\|v_A\|\| < 0.1$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
Sum of Instances	12	6	21	19	18	0	7	90	3	72	8

Table D.5: Overview of the squash vectors lengths for the remaining classes in comparison to the length of the squash vector for class 'N'.

Squash Vector 'N' Length Interval	**Largest Squash Vector of Non-Present Class**										
	A	B	C	E	K	O	Q	R	S	X	Z
	Max. length of largest squash vector of non-present class Min. length of largest squash vector of non-present class										
$0.9 \leq \|\|v_A\|\| < 1.0$	16	6	3	3	14	6	12	12	0	8	3
	0.28	0.14	0.14	0.14	0.16	0.28	0.21	0.17	-	0.27	0.16
	0.08	0.08	0.12	0.11	0.08	0.09	0.08	0.08	-	0.06	0.10
$0.8 \leq \|\|v_A\|\| < 0.9$	27	8	11	5	17	11	25	18	2	7	8
	0.31	0.21	0.28	0.23	0.47	0.45	0.36	0.33	0.28	0.36	0.20
	0.09	0.10	0.09	0.10	0.10	0.11	0.09	0.11	0.14	0.11	0.09
$0.7 \leq \|\|v_A\|\| < 0.8$	3	0	0	0	4	3	4	3	0	3	0
	0.53	-	-	-	0.44	0.33	0.45	0.24	-	0.42	-
	0.34	-	-	-	0.22	0.20	0.27	0.21	-	0.28	-
$0.6 \leq \|\|v_A\|\| < 0.7$	1	0	0	0	0	0	1	2	0	1	0
	0.28	-	-	-	-	-	0.47	0.35	-	0.39	-
	0.28	-	-	-	-	-	0.47	0.31	-	0.39	-
$0.5 \leq \|\|v_A\|\| < 0.6$	2	0	1	0	0	0	0	0	0	0	0
	0.33	-	0.61	-	-	-	-	-	-	-	-
	0.33	-	0.61	-	-	-	-	-	-	-	-
$0.4 \leq \|\|v_A\|\| < 0.5$	1	0	0	0	1	0	0	0	0	2	0
	0.43	-	-	-	0.52	-	-	-	-	0.35	-
	0.43	-	-	-	0.52	-	-	-	-	0.35	-
$0.3 \leq \|\|v_A\|\| < 0.4$	1	0	0	0	0	0	0	0	0	0	0
	0.59	-	-	-	-	-	-	-	-	-	-
	0.59	-	-	-	-	-	-	-	-	-	-
$0.2 \leq \|\|v_A\|\| < 0.3$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
$0.1 \leq \|\|v_A\|\| < 0.2$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
$0.0 \leq \|\|v_A\|\| < 0.1$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
Sum of Instances	51	14	15	8	36	20	42	35	2	21	11

Table D.6: Overview of the squash vectors lengths for the remaining classes in comparison to the length of the squash vector for class 'O'.

Squash Vector 'O' Length Interval	**Largest Squash Vector of Non-Present Class**										
	A	B	C	E	K	N	Q	R	S	X	Z
	Max. length of largest squash vector of non-present class Min. length of largest squash vector of non-present class										
$0.9 \leq \|\|v_A\|\| < 1.0$	10	16	11	3	5	9	8	9	6	3	0
	0.22	0.20	0.33	0.13	0.16	0.15	0.28	0.23	0.12	0.12	-
	0.07	0.08	0.10	0.11	0.09	0.09	0.10	0.11	0.05	0.09	-
$0.8 \leq \|\|v_A\|\| < 0.9$	15	17	32	10	3	8	21	13	5	1	5
	0.29	0.33	0.47	0.22	0.23	0.19	0.42	0.23	0.14	0.07	0.18
	0.08	0.10	0.09	0.12	0.10	0.10	0.09	0.10	0.09	0.07	0.12
$0.7 \leq \|\|v_A\|\| < 0.8$	2	4	8	1	0	2	14	2	2	0	2
	0.41	0.49	0.47	0.26	-	0.43	0.53	0.28	0.20	-	0.20
	0.21	0.16	0.12	0.26	-	0.21	0.14	0.16	0.17	-	0.20
$0.6 \leq \|\|v_A\|\| < 0.7$	0	1	0	0	0	0	2	0	0	0	0
	-	0.46	-	-	-	-	0.44	-	-	-	-
	-	0.46	-	-	-	-	0.37	-	-	-	-
$0.5 \leq \|\|v_A\|\| < 0.6$	1	1	0	0	0	0	1	0	0	0	0
	0.57	0.60	-	-	-	-	0.32	-	-	-	-
	0.57	0.60	-	-	-	-	0.32	-	-	-	-
$0.4 \leq \|\|v_A\|\| < 0.5$	0	1	0	0	0	0	0	0	0	0	0
	-	0.49	-	-	-	-	-	-	-	-	-
	-	0.49	-	-	-	-	-	-	-	-	-
$0.3 \leq \|\|v_A\|\| < 0.4$	0	0	1	0	0	0	0	0	0	0	0
	-	-	0.65	-	-	-	-	-	-	-	-
	-	-	0.65	-	-	-	-	-	-	-	-
$0.2 \leq \|\|v_A\|\| < 0.3$	0	0	1	0	0	0	0	0	0	0	0
	-	-	0.50	-	-	-	-	-	-	-	-
	-	-	0.50	-	-	-	-	-	-	-	-
$0.1 \leq \|\|v_A\|\| < 0.2$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
$0.0 \leq \|\|v_A\|\| < 0.1$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
Sum of Instances	28	40	53	14	8	19	46	24	13	4	7

Table D.7: Overview of the squash vectors lengths for the remaining classes in comparison to the length of the squash vector for class 'Q'.

Squash Vector 'Q' Length Interval	**Largest Squash Vector of Non-Present Class**										
	A	B	C	E	K	N	O	R	S	X	Z
	Max. length of largest squash vector of non-present class Min. length of largest squash vector of non-present class										
$0.9 \leq \|\|v_A\|\| < 1.0$	5	6	5	2	3	9	14	3	11	3	7
	0.11	0.23	0.16	0.23	0.14	0.18	0.30	0.18	0.33	0.18	0.27
	0.09	0.09	0.08	0.09	0.09	0.12	0.07	0.11	0.10	0.12	0.08
$0.8 \leq \|\|v_A\|\| < 0.9$	16	11	14	3	6	16	19	25	14	5	8
	0.34	0.36	0.39	0.38	0.21	0.25	0.34	0.44	0.46	0.22	0.26
	0.11	0.12	0.21	0.12	0.10	0.12	0.11	0.11	0.08	0.12	0.08
$0.7 \leq \|\|v_A\|\| < 0.8$	2	3	1	1	1	4	10	1	7	0	0
	0.41	0.30	0.41	0.22	0.17	0.54	0.67	0.37	0.39	-	-
	0.20	0.21	0.41	0.22	0.17	0.22	0.18	0.37	0.22	-	-
$0.6 \leq \|\|v_A\|\| < 0.7$	2	1	1	0	0	0	5	1	0	0	2
	0.52	0.38	0.38	-	-	-	0.65	0.49	-	-	0.26
	0.37	0.38	0.38	-	-	-	0.23	0.49	-	-	0.26
$0.5 \leq \|\|v_A\|\| < 0.6$	1	0	0	0	0	1	0	0	0	0	0
	0.41	-	-	-	-	0.28	-	-	-	-	-
	0.41	-	-	-	-	0.28	-	-	-	-	-
$0.4 \leq \|\|v_A\|\| < 0.5$	0	0	1	0	0	0	0	1	0	0	0
	-	-	0.43	-	-	-	-	0.61	-	-	-
	-	-	0.43	-	-	-	-	0.61	-	-	-
$0.3 \leq \|\|v_A\|\| < 0.4$	0	0	0	0	0	0	2	0	0	0	0
	-	-	-	-	-	-	0.79	-	-	-	-
	-	-	-	-	-	-	0.59	-	-	-	-
$0.2 \leq \|\|v_A\|\| < 0.3$	0	1	0	0	0	0	0	0	0	0	0
	-	0.38	-	-	-	-	-	-	-	-	-
	-	0.38	-	-	-	-	-	-	-	-	-
$0.1 \leq \|\|v_A\|\| < 0.2$	1	0	0	0	0	0	0	0	0	1	0
	0.49	-	-	-	-	-	-	-	-	0.31	-
	0.49	-	-	-	-	-	-	-	-	0.31	-
$0.0 \leq \|\|v_A\|\| < 0.1$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
Sum of Instances	27	22	22	6	10	30	50	31	32	9	17

Table D.8: Overview of the squash vectors lengths for the remaining classes in comparison to the length of the squash vector for class 'R'.

Squash Vector 'R' Length Interval	**Largest Squash Vector of Non-Present Class**										
	A	B	C	E	K	N	O	Q	S	X	Z
	Max. length of largest squash vector of non-present class Min. length of largest squash vector of non-present class										
$0.9 \leq \|\|v_A\|\| < 1.0$	1	0	0	4	2	1	0	0	0	2	0
	0.13	-	-	0.20	0.14	0.27	-	-	-	0.13	-
	0.13	-	-	0.11	0.14	0.27	-	-	-	0.10	-
$0.8 \leq \|\|v_A\|\| < 0.9$	10	13	1	11	59	10	2	12	3	8	13
	0.28	0.28	0.13	0.30	0.52	0.33	0.16	0.33	0.19	0.40	0.39
	0.09	0.10	0.13	0.09	0.12	0.12	0.12	0.10	0.09	0.12	0.11
$0.7 \leq \|\|v_A\|\| < 0.8$	4	7	2	8	17	5	0	9	1	6	4
	0.51	0.58	0.34	0.43	0.52	0.37	-	0.51	0.29	0.36	0.38
	0.14	0.23	0.25	0.19	0.16	0.18	-	0.19	0.29	0.15	0.25
$0.6 \leq \|\|v_A\|\| < 0.7$	4	3	4	4	4	3	0	3	0	1	0
	0.68	0.54	0.53	0.38	0.49	0.37	-	0.56	-	0.30	-
	0.24	0.26	0.38	0.21	0.26	0.22	-	0.39	-	0.30	-
$0.5 \leq \|\|v_A\|\| < 0.6$	0	1	2	0	2	0	0	2	0	0	0
	-	0.30	0.70	-	0.36	-	-	0.63	-	-	-
	-	0.30	0.43	-	0.30	-	-	0.57	-	-	-
$0.4 \leq \|\|v_A\|\| < 0.5$	1	1	0	0	2	0	0	0	0	0	0
	0.31	0.68	-	-	0.51	-	-	-	-	-	-
	0.31	0.68	-	-	0.40	-	-	-	-	-	-
$0.3 \leq \|\|v_A\|\| < 0.4$	0	1	0	0	0	0	0	1	0	0	0
	-	0.65	-	-	-	-	-	0.65	-	-	-
	-	0.65	-	-	-	-	-	0.65	-	-	-
$0.2 \leq \|\|v_A\|\| < 0.3$	2	0	0	0	0	0	0	0	0	0	0
	0.64	-	-	-	-	-	-	-	-	-	-
	0.61	-	-	-	-	-	-	-	-	-	-
$0.1 \leq \|\|v_A\|\| < 0.2$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
$0.0 \leq \|\|v_A\|\| < 0.1$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
Sum of Instances	22	26	27	9	86	19	2	27	4	17	17

Table D.9: Overview of the squash vectors lengths for the remaining classes in comparison to the length of the squash vector for class 'S'.

Squash Vector 'S' Length Interval	**Largest Squash Vector of Non-Present Class**										
	A	B	C	E	K	O	Q	R	S	X	Z
	Max. length of largest squash vector of non-present class Min. length of largest squash vector of non-present class										
$0.9 \leq \|\|v_A\|\| < 1.0$	4	21	2	12	7	4	2	10	18	7	7
	0.11	0.30	0.21	0.16	0.18	0.12	0.09	0.20	0.20	0.22	0.14
	0.08	0.11	0.11	0.10	0.07	0.08	0.09	0.08	0.08	0.09	0.08
$0.8 \leq \|\|v_A\|\| < 0.9$	1	34	5	23	4	7	10	19	7	13	9
	0.13	0.43	0.20	0.41	0.21	0.20	0.52	0.28	0.35	0.29	0.29
	0.13	0.10	0.13	0.08	0.12	0.09	0.12	0.07	0.10	0.08	0.11
$0.7 \leq \|\|v_A\|\| < 0.8$	1	4	1	4	0	0	5	5	0	3	0
	0.19	0.35	0.29	0.43	-	-	0.57	0.44	-	0.28	-
	0.19	0.17	0.29	0.20	-	-	0.16	0.17	-	0.22	-
$0.6 \leq \|\|v_A\|\| < 0.7$	0	2	1	0	0	0	0	0	0	1	0
	-	0.41	0.28	-	-	-	-	-	-	0.27	-
	-	0.33	0.28	-	-	-	-	-	-	0.27	-
$0.5 \leq \|\|v_A\|\| < 0.6$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
$0.4 \leq \|\|v_A\|\| < 0.5$	0	0	1	0	0	0	0	0	0	0	0
	-	-	0.47	-	-	-	-	-	-	-	-
	-	-	0.47	-	-	-	-	-	-	-	-
$0.3 \leq \|\|v_A\|\| < 0.4$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
$0.2 \leq \|\|v_A\|\| < 0.3$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
$0.1 \leq \|\|v_A\|\| < 0.2$	0	0	0	0	0	1	0	0	0	1	0
	-	-	-	-	-	0.54	-	-	-	0.46	-
	-	-	-	-	-	0.54	-	-	-	0.46	-
$0.0 \leq \|\|v_A\|\| < 0.1$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
Sum of Instances	6	61	10	39	11	12	17	34	25	25	16

Table D.10: Overview of the squash vectors lengths for the remaining classes in comparison to the length of the squash vector for class 'X'.

Squash Vector 'X' Length Interval	**Largest Squash Vector of Non-Present Class**										
	A	B	C	E	K	N	O	Q	R	S	Z
	Max. length of largest squash vector of non-present class Min. length of largest squash vector of non-present class										
$0.9 \leq \|\|v_A\|\| < 1.0$	10	2	11	4	13	4	0	3	2	5	3
	0.17	0.10	0.14	0.17	0.42	0.22	-	0.11	0.09	0.14	0.12
	0.07	0.07	0.08	0.07	0.09	0.10	-	0.08	0.09	0.09	0.08
$0.8 \leq \|\|v_A\|\| < 0.9$	12	3	6	10	53	12	1	9	15	12	20
	0.28	0.39	0.17	0.28	0.37	0.57	0.14	0.19	0.26	0.31	0.31
	0.09	0.09	0.07	0.07	0.09	0.07	0.14	0.08	0.09	0.08	0.08
$0.7 \leq \|\|v_A\|\| < 0.8$	5	0	0	1	12	1	0	0	3	1	3
	0.30	-	-	0.25	0.56	0.34	-	-	0.30	0.38	0.30
	0.14	-	-	0.25	0.16	0.34	-	-	0.25	0.38	0.30
$0.6 \leq \|\|v_A\|\| < 0.7$	0	1	0	0	4	0	0	0	0	0	1
	-	0.43	-	-	0.58	-	-	-	-	-	0.83
	-	0.43	-	-	0.34	-	-	-	-	-	0.83
$0.5 \leq \|\|v_A\|\| < 0.6$	0	0	0	0	5	0	0	1	0	2	0
	-	-	-	-	0.61	-	-	0.32	-	0.50	-
	-	-	-	-	0.32	-	-	0.32	-	0.50	-
$0.4 \leq \|\|v_A\|\| < 0.5$	0	0	0	0	0	0	1	0	0	0	0
	-	-	-	-	-	-	0.31	-	-	-	-
	-	-	-	-	-	-	0.31	-	-	-	-
$0.3 \leq \|\|v_A\|\| < 0.4$	0	0	0	0	1	0	0	0	0	1	0
	-	-	-	-	0.62	-	-	-	-	0.39	-
	-	-	-	-	0.62	-	-	-	-	0.39	-
$0.2 \leq \|\|v_A\|\| < 0.3$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
$0.1 \leq \|\|v_A\|\| < 0.2$	0	0	0	0	1	1	0	0	0	0	1
	-	-	-	-	0.48	0.68	-	-	-	-	0.50
	-	-	-	-	0.48	0.68	-	-	-	-	0.50
$0.0 \leq \|\|v_A\|\| < 0.1$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
Sum of Instances	27	6	17	15	89	18	2	13	20	21	28

Table D.11: Overview of the squash vectors lengths for the remaining classes in comparison to the length of the squash vector for class 'Z'.

Squash Vector 'Z' Length Interval	**Largest Squash Vector of Non-Present Class**										
	A	B	C	E	K	N	O	Q	R	S	X
	Max. length of largest squash vector of non-present class Min. length of largest squash vector of non-present class										
$0.9 \leq \|\|v_A\|\| < 1.0$	8	8	9	7	5	9	2	5	16	8	16
	0.12	0.16	0.21	0.16	0.16	0.14	0.13	0.33	0.25	0.15	0.25
	0.07	0.10	0.09	0.09	0.08	0.08	0.12	0.09	0.09	0.08	0.10
$0.8 \leq \|\|v_A\|\| < 0.9$	6	10	23	11	10	1	3	2	34	3	19
	0.26	0.31	0.45	0.34	0.39	0.17	0.19	0.15	0.40	0.16	0.37
	0.08	0.10	0.11	0.08	0.10	0.17	0.17	0.12	0.08	0.13	0.10
$0.7 \leq \|\|v_A\|\| < 0.8$	0	1	10	1	3	0	0	3	3	1	4
	-	0.55	0.53	0.27	0.41	-	-	0.33	0.39	0.30	0.40
	-	0.55	0.17	0.27	0.27	-	-	0.18	0.19	0.30	0.20
$0.6 \leq \|\|v_A\|\| < 0.7$	0	1	0	1	0	0	0	1	0	0	2
	-	0.43	-	0.39	-	-	-	0.21	-	-	0.37
	-	0.43	-	0.39	-	-	-	0.21	-	-	0.21
$0.5 \leq \|\|v_A\|\| < 0.6$	0	0	0	2	0	0	0	0	0	1	1
	-	-	-	0.60	-	-	-	-	-	0.65	0.49
	-	-	-	0.43	-	-	-	-	-	0.65	0.49
$0.4 \leq \|\|v_A\|\| < 0.5$	0	0	0	1	0	0	0	0	0	2	0
	-	-	-	0.34	-	-	-	-	-	0.37	-
	-	-	-	0.34	-	-	-	-	-	0.35	-
$0.3 \leq \|\|v_A\|\| < 0.4$	0	1	1	0	0	0	0	0	0	1	0
	-	0.65	0.65	-	-	-	-	-	-	0.47	-
	-	0.65	0.65	-	-	-	-	-	-	0.47	-
$0.2 \leq \|\|v_A\|\| < 0.3$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
$0.1 \leq \|\|v_A\|\| < 0.2$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
$0.0 \leq \|\|v_A\|\| < 0.1$	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	-	-	-	-
Sum of Instances	14	21	43	23	18	10	5	11	53	16	42

Image Number	A1	A2	A3	A4
Original				
Unmasked Prediction				
Masked Prediction				
$\|\|v_{present}\|\|$	$\|\|v_C\|\| = 0.89$	$\|\|v_C\|\| = 0.88$	$\|\|v_C\|\| = 0.83$	$\|\|v_C\|\| = 0.82$
Masked Prediction				
$\|\|v_{nonpresent}\|\|$	$\|\|v_Q\|\| = 0.23$	$\|\|v_A\|\| = 0.20$	$\|\|v_K\|\| = 0.23$	$\|\|v_O\|\| = 0.29$

Figure D.1: Comparison of masked predictions and unmasked predictions for class 'C', and the predictions for 'incorrect' squash vectors

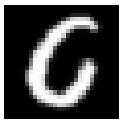
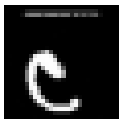
Image Number	A5	A6	A7	A8
Original				
Unmasked Prediction				
Masked Prediction				
$\|\|v_{present}\|\|$	$\|\|v_C\|\| = 0.75$	$\|\|v_C\|\| = 0.63$	$\|\|v_C\|\| = 0.63$	$\|\|v_C\|\| = 0.60$
Masked Prediction				
$\|\|v_{nonpresent}\|\|$	$\|\|v_Z\|\| = 0.37$	$\|\|v_O\|\| = 0.30$	$\|\|v_A\|\| = 0.48$	$\|\|v_K\|\| = 0.52$

Figure D.2: Comparison of masked predictions and unmasked predictions for class 'C', and the predictions for 'incorrect' squash vectors

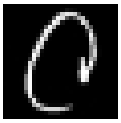
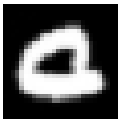
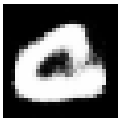
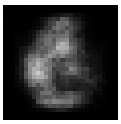
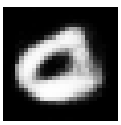
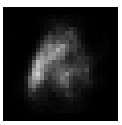
Image Number	A9	A10	A11	A12
Original				
Unmasked Prediction				
Masked Prediction				
$\|\|v_{present}\|\|$	$\|\|v_C\|\| = 0.51$	$\|\|v_C\|\| = 0.45$	$\|\|v_C\|\| = 0.44$	$\|\|v_C\|\| = 0.38$
Masked Prediction				
$\|\|v_{nonpresent}\|\|$	$\|\|v_R\|\| = 0.70$	$\|\|v_Q\|\| = 0.46$	$\|\|v_O\|\| = 0.76$	$\|\|v_O\|\| = 0.67$
Masked Prediction				
$\|\|v_{nonpresent}\|\|$	$\|\|v_N\|\| = 0.15$	$\|\|v_N\|\| = 0.29$	$\|\|v_N\|\| = 0.19$	$\|\|v_Q\|\| = 0.21$

Figure D.3: Comparison of masked predictions and unmasked predictions for class 'C', and the predictions for 'incorrect' squash vectors

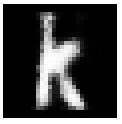
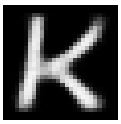
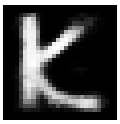
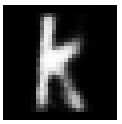
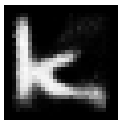
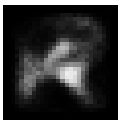
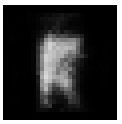
Image Number	A1	A2	A3	A4
Original				
Unmasked Prediction				
Masked Prediction				
$\|\|v_{present}\|\|$	$\|\|v_K\|\| = 0.90$	$\|\|v_K\|\| = 0.87$	$\|\|v_K\|\| = 0.85$	$\|\|v_K\|\| = 0.80$
Masked Prediction				
$\|\|v_{nonpresent}\|\|$	$\|\|v_R\|\| = 0.24$	$\|\|v_C\|\| = 0.62$	$\|\|v_R\|\| = 0.29$	$\|\|v_E\|\| = 0.60$

Figure D.4: Comparison of masked predictions and unmasked predictions for class 'K', and the predictions for 'incorrect' squash vectors

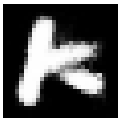
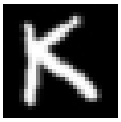
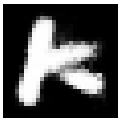
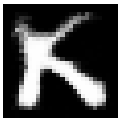
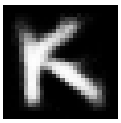
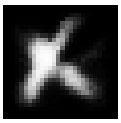
Image Number	A5	A6	A7	A8
Original				
Unmasked Prediction				
Masked Prediction				
$\|\|v_{present}\|\|$	$\|\|v_K\|\| = 0.77$	$\|\|v_K\|\| = 0.72$	$\|\|v_K\|\| = 0.68$	$\|\|v_K\|\| = 0.63$
Masked Prediction				
$\|\|v_{nonpresent}\|\|$	$\|\|v_Q\|\| = 0.38$	$\|\|v_X\|\| = 0.21$	$\|\|v_R\|\| = 0.31$	$\|\|v_X\|\| = 0.42$

Figure D.5: Comparison of masked predictions and unmasked predictions for class 'K', and the predictions for 'incorrect' squash vectors

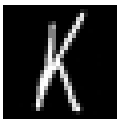
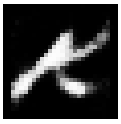
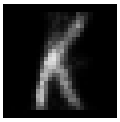
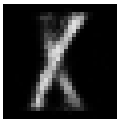
Image Number	A9	A10	A11	A12
Original				
Unmasked Prediction				
Masked Prediction				
$\|\|v_{present}\|\|$	$\|\|v_K\|\| = 0.57$	$\|\|v_K\|\| = 0.52$	$\|\|v_K\|\| = 0.44$	$\|\|v_K\|\| = 0.28$
Masked Prediction				
$\|\|v_{nonpresent}\|\|$	$\|\|v_R\|\| = 0.67$	$\|\|v_X\|\| = 0.52$	$\|\|v_C\|\| = 0.71$	$\|\|v_C\|\| = 0.38$

Figure D.6: Comparison of masked predictions and unmasked predictions for class 'K', and the predictions for 'incorrect' squash vectors

BIBLIOGRAPHY

[1] D. Lewandowski et al. "The influence of search engine optimization on Google's results: A multi-dimensional approach for detecting SEO". In: *13th ACM Web Science Conference 2021*. 2021, pp. 12–20.

[2] I. C. Drivas et al. "Big data analytics for search engine optimization". In: *Big Data and Cognitive Computing* 4.2 (2020), p. 5.

[3] M. P. Evans. "Analysing Google rankings through search engine optimization data". In: *Internet research* (2007).

[4] Y. Yuniarthe. "Application of artificial intelligence (AI) in search engine optimization (SEO)". In: *2017 International conference on soft computing, intelligent system and information technology (ICSIIT)*. IEEE. 2017, pp. 96–101.

[5] K. Sekaran et al. "Design of optimal search engine using text summarization through artificial intelligence techniques". In: *Telkomnika* 18.3 (2020), pp. 1268–1274.

[6] S. Das et al. "Applications of artificial intelligence in machine learning: review and prospect". In: *International Journal of Computer Applications* 115.9 (2015).

[7] S. Thiebes et al. "Trustworthy artificial intelligence". In: *Electronic Markets* 31.2 (June 2021), pp. 447–464. ISSN: 1422-8890. DOI: 10.1007/s12525-020-00441-4.

[8] P. Ala-Pietilä et al. *Ethics guidelines for trustworthy AI*. EU Publications, Apr. 2019. DOI: 10.2759/346720.

[9] J. Sharma et al. "Deep Convolutional Neural Networks for Fire Detection in Images". In: *Engineering Applications of Neural Networks*. Ed. by G. Boracchi et al. Cham: Springer International Publishing, 2017, pp. 183–193. ISBN: 978-3-319-65172-9.

[10] R. L. Galvez et al. "Object Detection Using Convolutional Neural Networks". In: *TENCON 2018 - 2018 IEEE Region 10 Conference*. 2018, pp. 2023–2027. DOI: 10.1109/TENCON.2018.8650517.

[11] T. Lei et al. "Landslide Inventory Mapping From Bitemporal Images Using Deep Convolutional Neural Networks". In: *IEEE Geoscience and Remote Sensing Letters* 16.6 (2019), pp. 982–986. DOI: 10.1109/LGRS.2018.2889307.

[12] L. Kang et al. “Convolutional Neural Networks for No-Reference Image Quality Assessment”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2014.

[13] R. R. Selvaraju et al. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 618–626. DOI: 10.1109/ICCV.2017.74.

[14] K. Simonyan et al. “Deep inside convolutional networks: Visualising image classification models and saliency maps”. In: *In Workshop at International Conference on Learning Representations*. Citeseer. 2014.

[15] M. T. Ribeiro et al. “” Why should i trust you?” Explaining the predictions of any classifier”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1135–1144.

[16] S. Sabour et al. “Dynamic Routing between Capsules”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 3859–3869. ISBN: 9781510860964.

[17] C. Gregory et al. “EMNIST: an extension of MNIST to handwritten letters”. In: *CoRR* abs/1702.05373 (2017). URL: http://arxiv.org/abs/1702.05373.

[18] A. Holzinger et al. “Current Advances, Trends and Challenges of Machine Learning and Knowledge Extraction: From Machine Learning to Explainable AI”. In: *Machine Learning and Knowledge Extraction*. Ed. by A. Holzinger et al. Springer International Publishing, 2018, pp. 1–8. ISBN: 978-3-319-99740-7.

[19] M. T. Ribeiro et al. “Anchors: High-precision model-agnostic explanations”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[20] Y. LeCun et al. “MNIST handwritten digit database”. In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[21] J. Springenberg et al. “Striving for Simplicity: The All Convolutional Net”. In: *ICLR (workshop track)*. 2015. URL: http://lmb.informatik.uni-freiburg.de/Publications/2015/DB15a.

[22] M. D. Zeiler et al. “Visualizing and understanding convolutional networks”. In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[23] K. H. Jin. "Deep Block Transform for Autoencoders". In: *IEEE Signal Processing Letters* 28 (2021), pp. 1016–1019.

[24] W. Yang et al. "Towards Rich Feature Discovery With Class Activation Maps Augmentation for Person Re-Identification". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 1389–1398. DOI: 10.1109/CVPR.2019.00148.

[25] M. Lin et al. *Network In Network*. 2014. arXiv: 1312.4400 [cs.NE].

[26] A. F. Agarap. "Deep learning using rectified linear units (relu)". In: *arXiv preprint arXiv:1803.08375* (2018).

[27] H. Geoffrey E. *Geoffrey Hinton: What is wrong with convolutional neural nets?* Sept. 26, 2017. URL: https://www.youtube.com/watch?v=Jv1VDdI4vy4.

[28] S. Birchfield. "Image Processing and Analysis". In: Cengage Learning, 2016, p. 165. ISBN: 9781337515627.

[29] V. Dumoulin et al. *A guide to convolution arithmetic for deep learning*. 2018. arXiv: 1603.07285 [stat.ML].

[30] S. Narayan. "The generalized sigmoid activation function: Competitive supervised learning". In: *Information Sciences* 99.1 (1997), pp. 69–82. ISSN: 0020-0255. DOI: https://doi.org/10.1016/S0020-0255(96)00200-9. URL: https://www.sciencedirect.com/science/article/pii/S0020025596002009.

[31] F. Chollet et al. *Keras*. https://github.com/fchollet/keras. 2015.

[32] B. Xu et al. "Empirical Evaluation of Rectified Activations in Convolutional Network". In: *CoRR* abs/1505.00853 (2015). arXiv: 1505.00853. URL: http://arxiv.org/abs/1505.00853.

[33] F. Chollet et al. *Lambda layer*. June 1, 2021. URL: https://keras.io/api/layers/core_layers/lambda/.

[34] M. Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. tf.reshape*. July 22, 2021. URL: https://www.tensorflow.org/api_docs/python/tf/reshape.

[35] M. Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. tf.keras.layers.Layer*. July 28, 2021. URL: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Layer.

[36] M. Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. tf.tile*. July 27, 2021. URL: https://www.tensorflow.org/api_docs/python/tf/tile.

[37] M. Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. tf.stack*. July 27, 2021. URL: https://www.tensorflow.org/api_docs/python/tf/stack.

[38] M. Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. tf.expand_dims*. July 23, 2021. URL: https://www.tensorflow.org/api_docs/python/tf/expand_dims.

[39] M. Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. tf.linalg.matmul*. July 23, 2021. URL: https://www.tensorflow.org/api_docs/python/tf/linalg/matmul.

[40] M. Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. tf.norm*. May 28, 2021. URL: https://www.tensorflow.org/api_docs/python/tf/norm.

[41] M. Abadi et al. *Eager execution — TensorFlow Core v2.5.0*. June 6, 2021. URL: https://www.tensorflow.org/guide/eager?hl=en.

[42] M. Abadi et al. *Introduction to graphs and tf.function — TensorFlow Core v2.5.0*. June 6, 2021. URL: https://www.tensorflow.org/guide/intro_to_graphs?hl=en.

[43] M. Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. tf.io.write_file*. July 11, 2021. URL: https://www.tensorflow.org/api_docs/python/tf/io/write_file.

[44] M. Abadi et al. *tf.keras.optimizers.schedules.ExponentialDecay*. Aug. 2, 2021. URL: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/schedules/ExponentialDecay.

[45] Z. Wang et al. "Image quality assessment: from error visibility to structural similarity". In: *IEEE transactions on image processing* 13.4 (2004), pp. 600–612.

[46] A. Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.